Validation fonctionnelle de contrôleurs logiques : contribution au test de conformité et à l'analyse en boucle fermée

Anaïs Guignard

EDSP École Doctorale Sciences Pratiques

ED N°285

THESE DE DOCTORAT
DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

présentée par

Mademoiselle Anaïs GUIGNARD

pour obtenir le grade de

DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Domaine :
Électronique, Électrotechnique, Automatique

Titre :

Validation fonctionnelle de contrôleurs logiques : contribution au test de conformité et à l'analyse en boucle fermée

Soutenue à Cachan, le 04 Décembre 2014, devant le jury composé de :

Thierry Jéron	Directeur de recherches, INRIA - SUMO	Président du jury
Bernard Riera	Professeur des universités., URCA - CReSTIC	Rapporteur
Zineb Simeu-Abazi	Maitre de conférences HDR, INPG - G-SCOP	Rapporteur
Laurent Piétrac	Maitre de conférences, INSA Lyon - AMPERE	Examinateur
Jean-Marc Faure	Professeur des universités, SUPMECA - LURPA	Directeur de thèse

Laboratoire Universitaire de Recherche en Production Automatisée
ENS de Cachan/ EA 1385/Université de Paris Sud 11
61 avenue du Président Wilson - 94235 Cachan Cedex

Remerciements

Ce manuscrit de thèse est le résultat de trois années de travail menées au sein du Laboratoire Universitaire de Recherche en Production Automatisée (LURPA) de l'Ecole Normale Supérieure de Cachan et plus précisément dans l'équipe ISA dirigée par Jean-Marc Roussel. Il n'existerait pas sous cette forme sans le concours de nombreuses personnes à qui je souhaite exprimer ma profonde gratitude.

Tout d'abord, je tiens à remercier Jean-Marc Roussel ainsi que l'ensemble du personnel enseignant de l'équipe ISA qui m'a accueillie et formée au cours de mes années d'études. C'est ce socle de connaissances solides qui m'a servi d'appui pour mener les travaux de recherche présentés dans ce manuscrit.

Je tiens aussi à remercier mon directeur de thèse, le Prof. Jean-Marc Faure, pour avoir encadré ma thèse tout au long de ces trois ans. Sa disponibilité, son écoute et son soutien ont été essentiels à la réussite de ces travaux et ce fut un honneur autant qu'un plaisir de travailler avec lui.

J'exprime aussi ma gratitude à mes rapporteurs Zineb Simeu-Abazi et le Prof. Bernard Riera pour leurs retours critiques et conseils grâce auxquels j'ai pu solidifier les dernières imprécisions restantes. Je remercie aussi Thierry Jéron et Laurent Piétrac pour avoir accepté de participer à mon jury de thèse ainsi que pour leurs questions variées.

Mais au-delà du manuscrit lui-même, un doctorat représente une longue et tortueuse épreuve qui ne peut être surmontée seul. Je souhaite donc remercier tous mes collègues, doctorants, enseignants, amis et famille qui ont su être à la fois de bon conseil et sources de détente. Ma gratitude va en particulier auprès de Jean-Marc Roussel, Pierre-Antoine, Julien et Jeremie. Ce doctorat aurait certainement été bien plus difficile à terminer sans l'existence du Civil et de NoLag. Enfin, mes derniers remerciements vont à Gregory qui a patiemment subi mes doutes et mes questions sans jamais faillir.

Résumé -

Les travaux présentés dans ce mémoire de thèse s'intéressent à la validation fonctionnelle de contrôleurs logiques par des techniques de test de conformité et de validation en boucle fermée. Le modèle de spécification est décrit dans le langage industriel Grafcet et le contrôleur logique est supposé être un automate programmable industriel (API) mono-tâche. Afin de contribuer à ces techniques de validation fonctionnelle, ces travaux présentent :

- Une extension d'une méthode de formalisation du Grafcet par traduction sous la forme d'une machine de Mealy. Cette extension permet de produire un modèle formel de la spécification lorsque le Grafcet est implanté selon un mode d'interprétation sans recherche de stabilité, qui n'est pas préconisé dans la norme IEC 60848 (2002) mais largement utilisé dans les applications industrielles.
- Une contribution au test de conformité par la définition d'un ensemble de relations de conformité basées sur l'observation de plusieurs cycles d'exécution pour chaque pas de test.
- Une contribution à la validation en boucle fermée par la définition d'un critère de fin d'observation et par une technique d'identification en boite grise pour la construction et l'analyse du système en boucle fermée.

Mots Clefs -

Validation fonctionnelle, Automate Programmable Industriel, Machine de Mealy, Test de conformité, Validation en boucle fermée, Modèles formels

Abstract -

The results presented in this PhD thesis deal with functional validation of logic controllers using conformance test and closed-loop validation techniques. The specification model is written in the Grafcet language and the logic controller is assumed to be a Programmable Logic Controller (PLC). In order to contribute to these validation techniques, this thesis presents :

- An axtension to a fomalization methods for Grafcet languages by translation to a Mealy machine. This extension generates a formal model of a Grafcet specification that is interpreted without search of stability. This mode of interpretation is not recommended by the standard IEC 60848 (2002) but is widely used in industrial applications.
- A contribution to conformance test by a definition of a set of conformance relation based on the observation of several execution cycles for each test step.
- A contribution to closed-loop validation by the definition of a termination criterion and by a new gray-box identification technique that is used for construction and analysis of the closed-loop system.

Keywords -

Functional validation, Programmable Logic Controller, Mealy machine, Conformance test, Closed-loop validation, Formal models

Table des matières

Table des figures

Introduction Générale

Les systèmes de contrôle/commande industriels d'aujourd'hui doivent répondre à des demandes de plus en plus importantes en terme d'automatisation des process et de sûreté. Afin de satisfaire ces demandes, de nombreux composants d'automatisation sont utilisés pour le contrôle tels que les calculateurs embarqués ou encore les cartes électroniques. Mais un dispositif largement utilisé pour des applications diverses telles que l'automobile, le transport ferroviaire ou encore la production d'électricité est le contrôleur logique de type **automate programmable industriel** (API). Ce type de contrôleur exécute un programme de commande qui sert à contrôler le comportement de la partie opérative, qui est le système à contrôler, en fonction des informations fournies par les divers capteurs associés à cette partie opérative.

Afin de garantir le bon fonctionnement des systèmes de contrôle/commande, il est impératif de s'assurer que le contrôleur agit bien comme le prévoit le cahier des charges. Pour répondre à ce besoin, des opérations ont été mises en place pendant le processus de conception pour garantir le niveau de sûreté exigé pour ce type de systèmes, aussi bien pendant la phase de développement de ces systèmes (méthodes hors ligne) que pendant leur phase d'opération (méthodes en ligne). Parmi les différentes opérations existantes (analyse de code, méthodes de synthèse, méthodes d'analyse formelle,...), l'une d'elles, qui servira d'objectif à l'ensemble des travaux développés dans ce mémoire de thèse, est la **validation fonctionnelle**. Elle a pour but s'assurer que le contrôleur logique programmable qui exécute le programme de commande se comporte comme indiqué dans la spécification.

C'est pourquoi le projet ANR VACSIM[1] dont le LURPA est le laboratoire coordinateur et qui regroupe trois autres partenaires académiques (I3S Sophia Antipolis, INRIA Rennes et LaBRI Bordeaux) et deux partenaires industriels (EDF R&D et Dassault Systemes) a été entrepris. En effet, ce projet consiste à tirer profit des avantages respectifs

1. Projet financé par l'Agence Nationale de la Recherche : Validation de la commande des systèmes critiques par couplage simulation et méthodes d'analyse formelle (ANR-11-INSE-004)

des techniques de simulation, en incluant des modèles des processus commandés, et des méthodes d'analyse formelles, pour la validation de la commande des systèmes critiques.

Cette thèse s'inscrit dans ce projet et vise plus précisément à contribuer à deux points particuliers de la validation fonctionnelle de contrôleurs logiques. En effet, de nombreuses techniques existent déjà pour la validation fonctionnelle et possèdent chacune leurs propres caractéristiques. Certaines sont basées sur un modèle du comportement du programme de commande implanté dans l'API, d'autres sont basées sur l'excitation des entrées de l'API, d'autres encore sur l'observation du contrôleur couplé à sa partie opérative. Le but d'une procédure de validation étant de garantir que le contrôleur contrôle la partie opérative en accord avec le cahier des charges, le choix est pris dans ces travaux de ne considérer que les techniques de validation basées sur l'excitation et/ou l'observation du comportement du contrôleur logique exécutant le programme de commande.

Les trois contributions de ce mémoire de thèse se situent dans la définition de **modèle formel de spécification** ainsi que dans le développement de techniques de validation fonctionnelle via des méthodes de **test de conformité** ou encore de **validation en boucle fermée**. Ces deux méthodes ont été choisies car elles présentent l'intérêt de valider non pas un modèle du contrôleur mais le contrôleur logique lui-même. Elles nécessitent néanmoins un modèle de spécification formellement défini qui décrive le comportement attendu de ce contrôleur logique.

Le test de conformité est une méthode de validation fonctionnelle qui requiert l'isolement du contrôleur logique par rapport à la partie opérative. En imposant les valeurs des variables d'entrée à ce contrôleur, il est possible d'observer la réaction du contrôleur et comparer cette réaction avec celle prévue par le modèle de spécification. Cette comparaison se fait via une relation de conformité généralement définie à partir d'un modèle du contrôleur logique. Ce mémoire de thèse propose de définir un ensemble de relations de conformité qui soit capable de prendre en compte des phénomènes propre à la technologie du contrôleur logique et qui conduisent les relations de conformité existantes à des verdicts biaisés (verdict de non-conformité pour un contrôleur logique conforme).

La validation en boucle fermée propose de placer le contrôleur logique dans une configuration d'utilisation finale, c'est-à-dire couplé à une partie opérative, et d'observer les évènements d'entrée et de sortie ayant lieu entre ce contrôleur et la partie opérative afin de les comparer avec la spécification. La partie opérative peut alors être réelle

ou simulée par un logiciel informatique afin de limiter les conséquences d'un éventuel comportement non-valide. Jusque là, cette configuration a été utilisée afin d'y appliquer des séquences d'évènements précises, souvent critiques, à valider. Ce mémoire de thèse propose une alternative qui permette de valider une part plus grande du comportement du contrôleur.

Ce mémoire de thèse se découpe alors en quatre chapitres qui exposent successivement le contexte scientifique des travaux, une solution pour la construction d'un modèle formel de spécification à partir d'une spécification Grafcet, une contribution aux techniques de test de conformité et enfin une contribution par une nouvelle technique de validation en boucle fermée.

Le premier chapitre permet de définir le contexte et les objectifs des travaux. Les principes de fonctionnement d'un contrôleur logique programmable ainsi que les principales techniques de validation fonctionnelle y sont rappelées. La mise en évidence de certaines limitations de ces techniques permet de poser les objectifs de nos travaux.

Le second chapitre présente la première contribution de ces travaux. Celle-ci traite de la construction du modèle formel d'une spécification à partir d'un modèle industriel de cette spécification. Ici, pour des raisons détaillées dans le second chapitre, le modèle issu du cahier des charges est choisi comme étant un Grafcet et le modèle formel comme étant une machine de Mealy. Cette construction a déjà été étudiée dans Provost et al. (2011b) mais s'est limité à un seul type d'interprétation du Grafcet. Ce chapitre étend donc ces résultats pour le second type d'interprétation et présente les différences entre les deux modèles obtenus en fonction de l'interprétation choisie.

Le troisième chapitre présente une contribution aux techniques de test de conformité. En effet, de nombreux travaux et résultats existent déjà sur ces techniques mais ceux-ci supposent systématiquement une lecture correcte par le contrôleur des entrées appliquées. Or, des études expérimentales ont déjà montré que certains phénomènes inhérents à la technologie et donc inévitables pouvaient conduire à une mauvaise lecture des entrées envoyées. Après une première extension des travaux présentés dans Provost et al. (2011a) sous la forme d'une définition formelle d'une relation de conformité lorsque les entrées sont lues correctement par le contrôleur, un second ensemble de relations de conformité est proposé afin de prendre en compte les erreurs de lecture des entrées.

Le quatrième chapitre présente, lui, une nouvelle technique de validation en boucle

fermée. En effet, bien que le test de conformité soit une technique qui assure la validité de la totalité du comportement du contrôleur, celui-ci est testé isolé de sa partie opérative. Si l'on souhaite garantir un verdict qui soit valable pour le système de contrôle/commande en conditions d'utilisation, il est nécessaire d'effectuer la validation lorsque le contrôleur est couplé à sa partie opérative. Ce chapitre propose donc comme contribution une solution pour ce type de technique de validation depuis la phase d'observation jusqu'au verdict. Les divers phénomènes dus aux erreurs de lecture des entrées seront aussi étudiés.

En conclusion, une synthèse des résultats obtenus lors de cette thèse est présentée et des perspectives pour des travaux futurs sont proposées.

Chapitre 1
Contexte et objectifs des travaux

Sommaire

Introduction

Le travail présenté dans ce mémoire de thèse s'intéresse aux techniques de validation fonctionnelle de contrôleurs logiques industriels. Dans ce chapitre, le contexte ainsi que les objectifs de ces travaux sont présentés. Il est structuré comme suit :

- La première section décrit brièvement la structure et le fonctionnement d'un automate programmable industriel (API).
- La seconde section présente le principe de la vérification et de la validation ainsi que l'objectif global des travaux présentés dans ce mémoire.
- La troisième section définit formellement les modèles utilisés dans ces travaux.
- La quatrième section présente diverses méthodes d'analyse ainsi qu'une argumentation sur le choix des méthodes utilisées dans ces travaux.
- Enfin, la dernière section énonce les différentes contributions présentées dans ce mémoire pour la validation fonctionnelle de contrôleurs logiques.

1.1 Les automates programmables industriels (API)

Les systèmes de contrôle/commande sont des systèmes ayant pour objectif de réaliser une ou plusieurs fonctions de commande séquentielles ou continues, ainsi que la communication via des interfaces homme-machine. La Figure 1.1 présente une architecture de contrôle-commande où les différents niveaux apparaissent clairement. Un ensemble de capteurs, de détecteurs, d'actionneurs et de pré-actionneurs, composant la partie opérative, est relié à un ensemble d'automates. A partir des informations envoyées par les capteurs et des ordres du niveau supervision, ces automates vont contrôler la partie opérative en envoyant des ordres aux pré-actionneurs et actionneurs.

Dans le système de contrôle/commande, ces travaux se focalisent sur la partie automate en charge des fonctions séquentielles, composée de contrôleurs logiques et plus précisément d'automates programmables industriels, très largement répandus dans les installations industrielles. Les différents éléments présentés dans la suite de cette section le sont en accord avec la norme IEC 61131 (IEC 61131-1..IEC 61131-8 (2000-2010)) qui

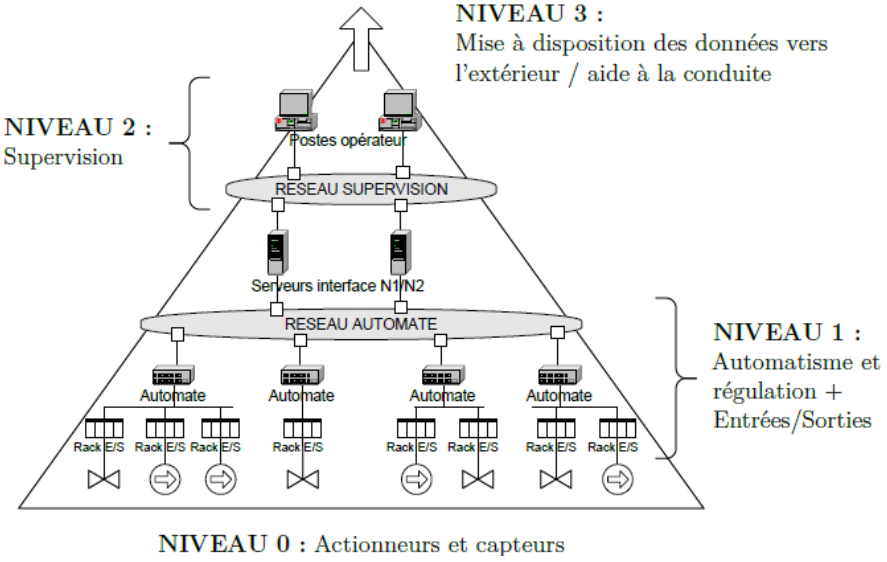


Figure 1.1 – Un exemple d'architecture de système de contrôle-commande (source : EDF R&D)

définit les diverses règles de conception, de fonctionnement, de programmation et de validation des API.

1.1.1 Architecture matérielle

Un API peut être composé sous la forme modulaire dont la structure est présentée en Figure 1.2. Il est composé d'un module de base, de modules d'entrée et de modules de sortie. Les différentes fonctions associées à chaque modules sont décrites ci-après.

Le module de base doit réaliser trois fonctions distinctes :

- Stocker le programme embarqué qui exécute la logique de commande ainsi que les valeurs des variables d'entrée, de sortie, et internes ; cette fonction est réalisée à l'aide de différentes mémoires.
- Exécuter le programme embarqué qui exécute la logique de commande ; cette fonction est réalisée à l'aide d'un processeur.
- Communiquer avec les modules d'entrée et de sortie ainsi qu'avec l'interface de programmation ; cette fonction est réalisée à l'aide de divers sous-modules de communication.

Les modules d'entrée, ou cartes d'entrée, ont pour fonction de recevoir les informations issues des capteurs de la partie opérative et de transmettre les valeurs au module

de base. Ces valeurs sont de type logique pour un API. Chaque module peut traiter un nombre fini de variables d'entrée, souvent regroupées par type de variables afin de faciliter le traitement des données.

Les modules de sortie, ou cartes de sortie, ont pour fonction de recevoir les ordres envoyés par le programme embarqué à transmettre aux pré-actionneurs de la partie opérative. De même que pour les modules d'entrée, les variables de sortie sont de type logique dans le cas d'un API.

Les modules d'entrée et de sortie peuvent aussi être déportés. Les variables transmises et reçues par le module de base transitent alors via des réseaux de communication. Cette solution est parfois nécessaire dans le cas de système de contrôle-commande de grande taille.

1.1.2 Cycle d'exécution de la logique de commande

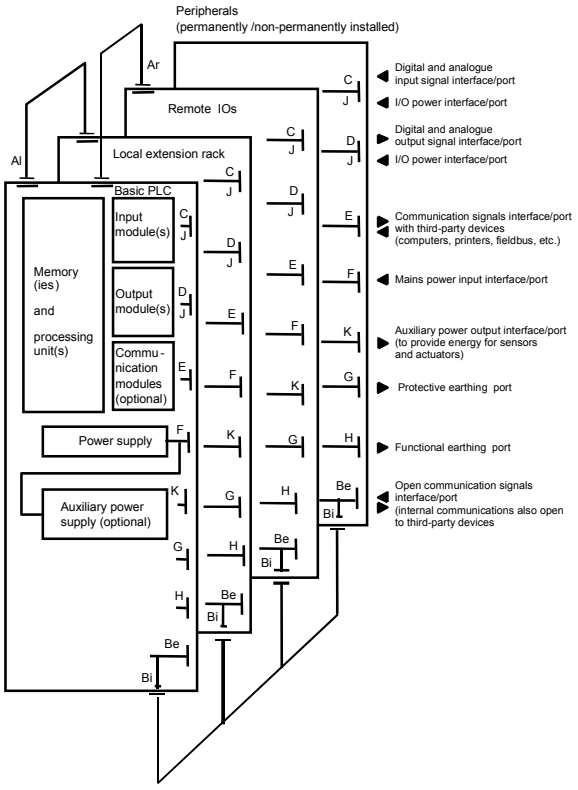
Le fonctionnement d'un API consiste en l'exécution séquentielle de plusieurs opérations contrôlées par un moniteur d'exécution. Ce moniteur d'exécution temps-réel peut être mono-tâche ou multi-tâches. Dans le cadre de ces travaux, **seul le cas d'un API avec un moniteur d'exécution mono-tâche est considéré.**

Ces opérations sont :

- Lecture (ou scrutation) des entrées : Les modules d'entrée sont contactés afin de récupérer les informations sur les capteurs de la partie opérative. Ces informations, sous forme de variables logiques, sont lues et stockées dans une table de variables. Les valeurs dans cette table des variables sont maintenues jusqu'à la prochaine lecture des entrées.

- Traitement du programme utilisateur : Le programme embarqué est exécuté une fois. A partir des valeurs des variables d'entrée stockées dans la table des variables, il calcule l'ensemble des valeurs des variables internes et de sortie qu'il stocke dans la table des variables.

- Ecriture (ou mise à jour) des sorties : Les valeurs des variables de sortie calculées par le programme embarqué sont envoyées, via les modules de sortie, aux pré-actionneurs de la partie opérative. La valeur de ces variables sera maintenue jusqu'à la prochaine écriture des sorties.

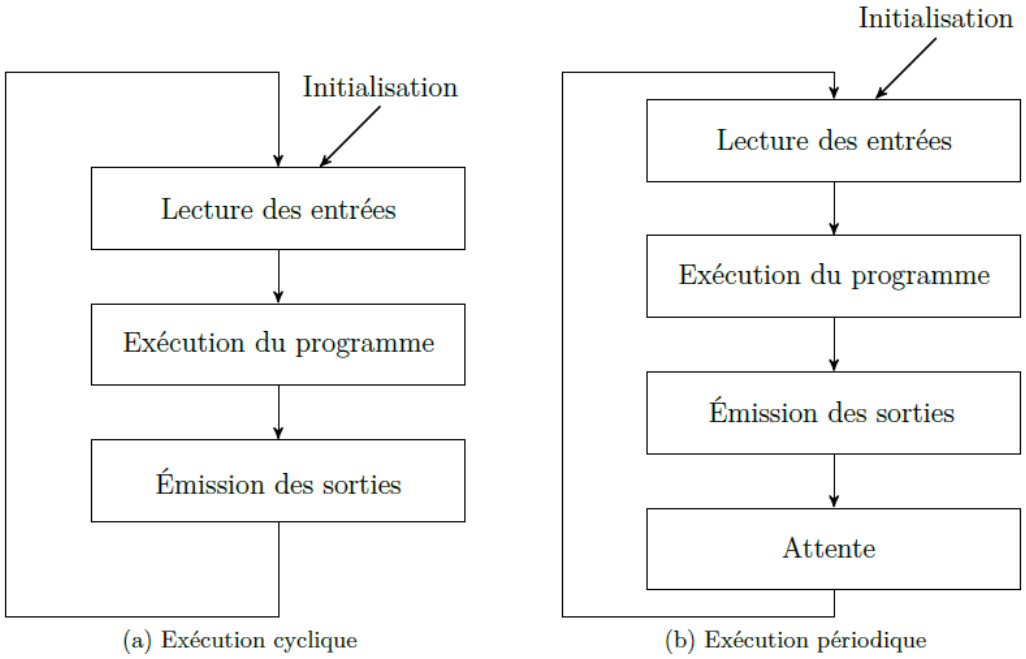


Key

Al	Communication interface/port for local I/O
Ar	Communication interface/port for remote I/O station
Be	Open-communication interface/port also open to third-party devices (for example, personal computer used for programming instead of a PADT)
Bi	Internal communication interface/port for peripherals
C	Interface/port for digital and analogue input signals
D	Interface/port for digital and analogue output signals
E	Serial or parallel communication interfaces/ports for data communication with third-party devices
F	Mains power interface/port. Devices with F ports have requirements on keeping downstream devices intelligent during power-up, power-down and power interruptions.
G	Port for protective earthing
H	Port for functional earthing
J	I/O power interface/port used to power sensors and actuators
K	Auxiliary power output interface/port

Figure 1.2 – Structure d'un API modulaire issue de IEC 61131-2 (2007)

Pour ce type de moniteur d'exécution, deux modes de fonctionnement peuvent être utilisés pour l'exécution séquentielle des différentes opérations : cyclique ou périodique. En mode cyclique, l'API va exécuter les opérations puis revenir au début de son cycle d'exécution dès la fin du cycle précédent. En mode périodique, chaque cycle dure un temps défini ; un temps d'attente doit donc être observé après l'écriture des sorties jusqu'au début d'un nouveau cycle. Ainsi, le mode cyclique garantit une meilleure réactivité du système alors que le mode périodique assure une certaine régularité dans la mise à jour des variables de sortie. La Figure 1.3 récapitule les différentes opérations effectuées en fonction du mode d'exécution de l'automate.



(a) Exécution cyclique (b) Exécution périodique

Figure 1.3 – Cycles du moniteur d'exécution d'un API

1.1.3 Phénomènes notables lors de l'exécution du cycle

Comme cela a été décrit précédemment, l'exécution du cycle d'un API passe par plusieurs opérations distinctes. Cette succession d'opérations peut avoir des conséquences sur la manière dont sont perçus les changements de valeurs des variables d'entrée. Cette sous-section présente quels sont les phénomènes qui peuvent en résulter. **Les différents travaux développés dans ce mémoire auront notamment pour but d'être capable de prendre en compte ces différents phénomènes**.

On peut remarquer que la lecture des entrées ne se fait pas tout au long du cycle mais seulement à son début. Cela signifie alors que l'API ne suit pas en temps réel les valeurs

des variables d'entrée mais procède à des lectures ponctuelles de ces valeurs. De plus, la lecture des variables d'entrée est une opération informatique qui n'est pas instantanée car elle requiert la copie, une à une, des valeurs lues dans les modules d'entrée vers la table des variables. Bien qu'une copie locale soit effectuée dans chaque module d'entrée, ceux-ci ne sont pas synchronisés et des écarts de temps entre la lecture de deux variables d'entrée peuvent exister.

Ces deux aspects du fonctionnement d'un cycle API vont avoir trois conséquences sur la lecture des valeurs des variables d'entrée par l'API. En nommant un changement de valeur d'une variable d'entrée un évènement, ces trois phénomènes sont :

- le retard d'un cycle automate entre l'occurrence d'un évènement d'entrée et sa lecture.
- la lecture synchrone d'évènements asynchrones, aussi appelé synchronisation des variables d'entrée,
- la lecture asynchrone d'évènements synchrones, aussi appelé désynchronisation des variables d'entrée.

Il est supposé dans l'ensemble de ces travaux que ces phénomènes ne se produisent qu'au niveau des entrées logiques du contrôleur. Les sorties logiques, elles, seront toujours observées telles qu'émises par le contrôleur.

1.1.3.1 Retard à la lecture des entrées

Le phénomène de retard à la lecture d'évènement est illustré par la Figure 1.4. Il correspond au fait qu'un évènement d'entrée (passage à $Vrai$ de la variable I_1) qui se produit au cycle i est perçu au mieux au cycle $i+1$.

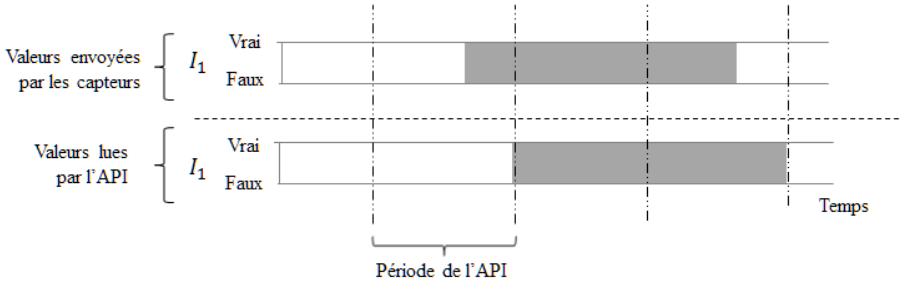


Figure 1.4 – Phénomène de retard à la lecture d'évènement

1.1.3.2 Lecture synchrone d'évènements asynchrones

Le phénomène de lecture synchrone d'évènements asynchrones est le phénomène le plus répandu et le plus connu (Fabian and Hellgren (1998)). La Figure 1.5 illustre ce phénomène. Il est dû à la nature même de la lecture cyclique des variables d'entrée. Ainsi, si deux variables d'entrée changent de valeur durant un cycle API à des instants différents (passage à $Vrai$ de la variable I_1 puis passage à $Vrai$ de la variable I_2), l'API peut percevoir ces changements de valeurs comme simultanés.

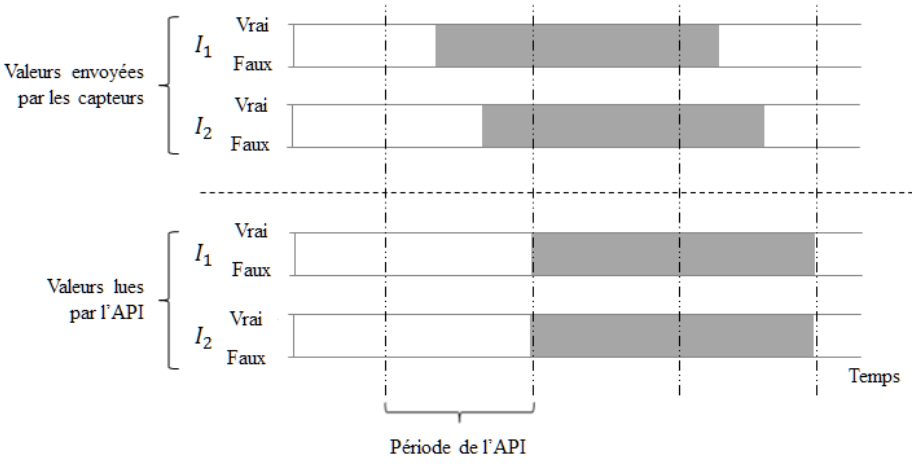


Figure 1.5 – Phénomène de lecture synchrone d'évènements asynchrones

1.1.3.3 Lecture asynchrone d'évènements synchrones

Le phénomène de lecture asynchrone d'évènements synchrones est un phénomène nettement moins courant lors de l'exécution du cycle automate. La Figure 1.6 illustre ce phénomène. Il est dû au fait que les variables d'entrée ne sont pas lues de manière synchronisée. Ainsi, une partie des évènements d'entrée peut être perçue seulement au cycle API suivant (ici, le passage à $Vrai$ de la variable I_2).

1.1.3.4 Retour sur le retard à la lecture des entrées

Le phénomène de lecture asynchrone d'évènements synchrones met en évidence le fait que si une variable d'entrée change de valeur à un instant proche du début de la phase de lecture des entrées par l'API, ce changement peut ne pas être perçu par l'API. Lorsque seule une partie des évènements ne sont pas perçus, cela mène au phénomène précédent, mais lorsque la totalité des évènements ne sont pas perçus lors de la lecture des entrées, cela revient à un retard d'au plus deux cycles API sur la lecture des entrées.

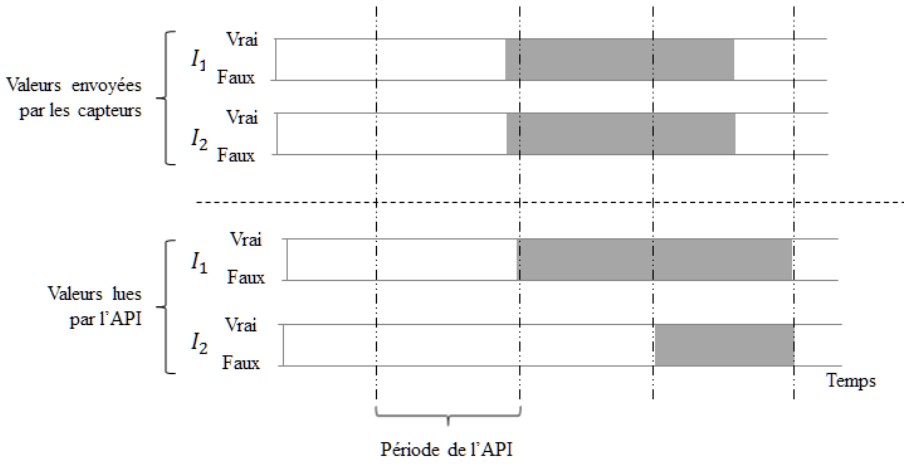


Figure 1.6 – Phénomène de lecture asynchrone d'évènements synchrones

La Figure 1.7 en propose une illustration lorsque seul un évènement a lieu.

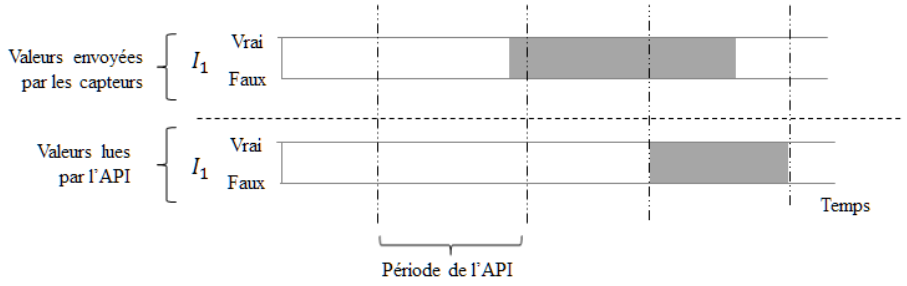


Figure 1.7 – Phénomène de retard à la lecture d'évènement

1.2 Vérification et validation

Lors de la conception d'un produit, ici d'un système de contrôle/commande, chaque étape nécessite des phases d'évaluation des opérations réalisées. Cela est illustré par la Figure 1.8.

Chacune de ces évaluations peut se faire dans un but de validation ou de vérification. Ces deux termes, définis dans la norme IEC 1012 (2004), sont synthétisés dans Boehm (1979) par la phrase suivante : la vérification est une méthode qui permet de s'assurer que l'on fait bien le produit, tandis que la validation est une méthode qui permet de s'assurer que **l'on fait le bon produit**. D'un point de vue pratique, la réalisation d'opérations de vérification et de validation peuvent s'illustrer selon la Figure 1.9. Les travaux présentés dans ce mémoire se focalisent sur les opérations de validation.

Les différentes opérations de validation peuvent se répertorier plus finement en fonction de leur objectif. Ainsi, les différentes techniques de validation s'organisent en 4

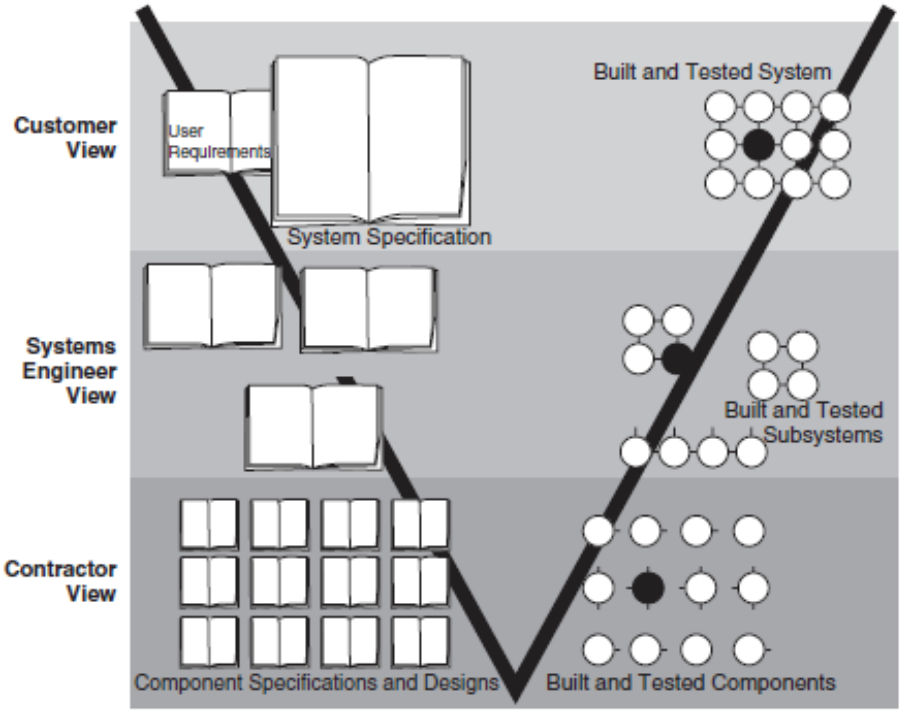


Figure 1.8 – Cycle en V selon Patterson (2009)

catégories :

- La validation des processus qui vise à produire des preuves qu'un processus est capable de produire un produit respectant les exigences qualité.
- La validation structurelle qui s'intéresse à la structure du système à valider.
- La validation fonctionnelle qui détermine si les fonctions spécifiées sont correctement réalisées par le système à valider.
- La validation de performances qui s'intéresse aux performances temporelles d'un système.

L'objectif premier des travaux présentés dans ce mémoire est de garantir le bon comportement des systèmes de contrôle/commande. Pour cela, nous nous focalisons sur la partie contrôle et en particulier sur le contrôleur logique en charge d'exécuter le programme logique qui commande la partie opérative. Cela revient donc à chercher à déterminer si un programme de commande implanté dans un API agit bien conformément à sa spécification. C'est pour cela que la suite de ce mémoire traitera de **techniques de validation fonctionnelle de contrôleurs logiques**.

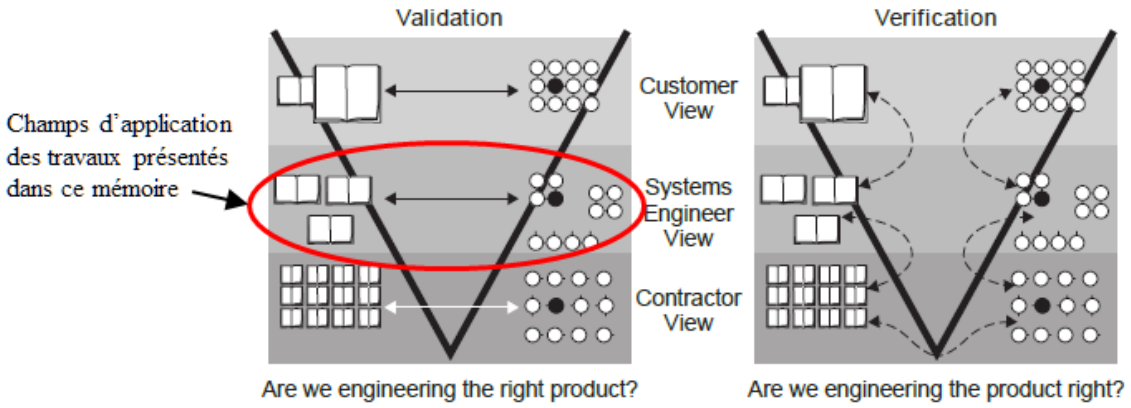


Figure 1.9 – Validation et vérification dans le cycle de développement (Patterson (2009))

La section suivante présente quels sont les modèles choisis sur lesquels devront être appliquées les techniques de validation fonctionnelle.

1.3 Modèles utilisés dans ces travaux

Une opération de validation fonctionnelle de contrôleur logique consiste à s'assurer que le programme de commande implanté dans le contrôleur agit bien en accord avec la spécification. Cela signifie qu'il faut avoir un modèle de spécification qui décrive le comportement attendu. Ce modèle est construit à partir du cahier des charges décrivant les diverses propriétés que doit vérifier le système de contrôle/commande final. Il doit être capable de traiter des variables d'entrée et de sortie et être utilisé dans le domaine industriel afin de rendre les techniques développées utilisables par les industriels. Dans ces travaux, il est supposé que **le modèle représentant la spécification est de type Grafcet**.

D'autre part, il est souhaité dans ces travaux de rester **non-invasif**. Cela signifie qu'il est exclu d'aller chercher des informations à l'intérieur de l'API. L'API est donc vue comme une boite noire dans laquelle le programme de commande est inconnu et où seules les valeurs des variables échangées avec l'extérieur sont observables. Ainsi, seules les variables d'entrée et de sortie de l'API peuvent être utilisées pour réaliser l'opération de validation fonctionnelle.

Cependant, la validation fonctionnelle est une opération de comparaison entre un comportement d'un contrôleur et celui de sa spécification. Il est donc nécessaire de pouvoir modéliser de manière formelle ces deux comportements pour pouvoir les comparer.

Ce problème apparait déjà dans Lhoste et al. (1993) et Bierel et al. (1997) mais les solutions proposées, sous la forme d'un méta-modèle statique ou de description algorithmique ne sont pas adaptées pour la validation fonctionnelle. La traduction du modèle Grafcet en un autre modèle qui soit formel est donc privilégiée. Ce modèle formel devra alors représenter soit le comportement attendu, issu du cahier des charges, soit le comportement du programme de commande, construit à partir d'observations. Ce modèle doit être capable de traiter de variables d'entrée et de sortie et avoir une structure adaptée à l'analyse formelle. Dans ces travaux, **le modèle formel choisi est la machine de Mealy**, type de modèle déjà utilisé pour la validation fonctionnelle et notamment pour le test de conformité (Lee and Yannakakis (1996)).

A partir de ce modèle formel, il est aussi possible de définir quels seront les types de faute détectables par les techniques de validation fonctionnelle développées.

1.3.1 Modèle du comportement spécifié dans le cahier des charges : le Grafcet

Le modèle de spécification issu du cahier des charges est supposé être un modèle Grafcet (David and Alla (2007)) tel que défini dans la norme IEC 60848 (2002). Une définition formelle de ce modèle de spécification, issue de Provost et al. (2011b) est construite comme un $6-uplet$ $(V_I, V_O, S_G, T_G, A_G, S_{Init})$ où :

- V_I est l'ensemble fini non vide des entrées du système logique (ici, les variables associées aux capteurs),
- V_O est l'ensemble fini non vide des sorties du système logique (ici, les variables associées aux pré-actionneurs),
- S_G est l'ensemble non vide des étapes du Grafcet, avec X_G l'ensemble de variables d'activité associées[2],
- T_G est l'ensemble non vide des transitions du Grafcet,
- A_G est l'ensemble des actions du Grafcet,
- S_{Init} est l'ensemble des étapes initiales du Gracet.

Une transition $t \in T_G$ est définie par un $3-uplet$ $(S_U, S_D, FC(V_I, X_G))$ où :

- S_U est l'ensemble des étapes amont,
- S_D est l'ensemble des étapes aval,
- $FC(V_I, X_G)$ est la réceptivité associée à la transition, expression booléenne dépendant des variables d'entrée et des variables d'activité d'étape où l'opérateur . représente la conjonction, + représente la disjonction et $\bar{}$ représente le complément.

L'ensemble des actions A_G est partitionné en deux sous-ensembles :

- L'ensemble des actions continues A_C. Ces actions sont dites continues car la sortie associée à chacune d'elles n'est émise que lorsqu'une étape portant l'action est active. De plus, une expression booléenne dépendant des variables d'entrée et des variables d'activité d'étape peut être associée à une étape portant l'action. Dans ce cas, l'action est dite conditionnelle et ne sera émise que si l'étape est active et que l'expression booléenne est $Vrai$.
- L'ensemble des actions mémorisées A_S. Ces actions sont dites mémorisées car le début ou l'arrêt de l'émission de la sortie associée à une telle action se fait en fonction d'ordres de type *Set* et *Reset* générés lorsque les étapes portant l'action sont actives.

La structure de ce modèle définie, certaines règles d'évolution doivent aussi être respectées selon la norme IEC 60848 (2002) :

- Règle n°1 : La situation initiale choisie par le concepteur, est la situation, caractérisée par l'ensemble des étapes actives, à l'instant initial.
- Règle n°2 : Une transition est dite validée lorsque toutes les étapes immédiatement précédentes reliées à cette transition sont actives. Le franchissement d'une transition se produit lorsque la transition est validée et que la réceptivité associée à cette transition est vraie.

2. Pour chaque étape $s \in S_G$, la variable d'activité de cette étape $x \in X_G$ vaut $Vrai$ si l'étape est active, $Faux$ sinon.

- Règle n°3 : Le franchissement d'une transition entraîne simultanément l'activation de toutes les étapes immédiatement suivantes et la désactivation de toutes les étapes immédiatement précédentes.

- Règle n°4 : Plusieurs transitions simultanément franchissables sont simultanément franchies.

- Règle n°5 : Si, au cours du fonctionnement, une étape active est simultanément activée et désactivée, alors elle reste active.

Un Grafcet peut être composé d'un unique ou de plusieurs graphes indépendants contenant chacun au moins une étape initiale. D'autres éléments sont aussi définis dans la norme. On retrouve notamment les macro-étapes qui sont une manière condensée de représenter une portion de graphe, les ordres de forçage qui déclenchent l'activation ou la désactivation d'étapes et les étapes encapsulantes qui déclenchent l'activation et la désactivation de graphes.

Quelques hypothèses sont retenues sur le modèle de spécification pour ces travaux :

- Le modèle de spécification ne contient aucun élément temporisé (temporisation sur les transitions, actions retardées ou limitées en temps).

- Les réceptivités associées aux transitions ne contiennent ni front montant ni front descendant sur des variables logiques.

- Le modèle de spécification ne contient aucune étape encapsulante ou ordre de forçage.

- Le modèle de spécification ne contient aucune étape source ou puits.

La Figure 1.10 propose un exemple de modèle de spécification contenant 2 variables d'entrée $V_I = \{a, b\}$ et 2 variables de sortie $V_O = \{O_1, O_2\}$ vérifiant ces hypothèses. Il est contient un seul graphe composé de 5 étapes $S_G = \{1, 2, 3, 4, 5\}$ dont l'étape $S_{Init} = 1$ et de 4 transitions $T_G = \{t_1, t_2, t_3, t_4\}$. Des actions continues sont associées aux étapes 1 et 4 et une action conditionnelle est associée à l'étape 5 qui n'émet la sortie O_2 que lorsque l'étape 4 est inactive.

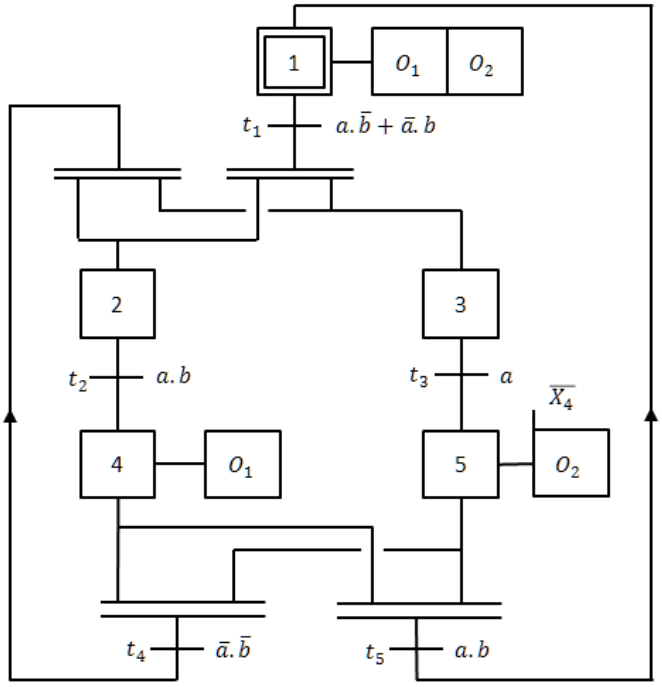


Figure 1.10 – Exemple simple de Grafcet

1.3.2 Modèle formel pour l'analyse : la machine de Mealy

1.3.2.1 Définition

Formellement, une machine de Mealy est définie par un 6-uplet (I_M, O_M, S_M, s_{init}, δ_M, λ_M) tel que :

- I_M est un alphabet fini d'entrée,
- O_M est un alphabet fini de sortie,
- S_M est un ensemble fini d'états,
- $s_{init} \in S_M$ est l'état initial,
- $\delta_M : I_M \times S_M \rightarrow S_M$ est la fonction de transition,
- $\lambda_M : I_M \times S_M \rightarrow O_M$ est la fonction de sortie.

1.3.2.2 Cas d'une machine de Mealy décrivant un système logique

Un contrôleur logique lit les valeurs d'un ensemble de variables Booléennes V_I. Chaque élément de l'alphabet d'entrée I_M représente une combinaison de valeurs des variables d'entrée. Il est défini par la valeur de chacune des variables d'entrée et est représenté sous la forme d'une combinaison Booléenne (appelée minterme) sur ces variables

d'entrée. Plus précisément, cette combinaison Booléenne est obtenue par la conjonction de toutes les variables d'entrée sous leur forme complémentée ou non-complémentée. La taille de l'alphabet d'entrée vaut alors $|I_M| = 2^{|V_I|}$

A partir de l'exemple de modèle de spécification présenté en Figure 1.10, l'ensemble des variables d'entrée est $V_I = \{a, b\}$. L'alphabet d'entrée de la machine de Mealy est alors composé de toutes les combinaisons Booléennes possibles à partir de ces variables d'entrée $I_M = \{\overline{a}.\overline{b}, a.\overline{b}, \overline{a}.b, a.b\}$. Cet alphabet d'entrée contient bien $2^2 = 4$ éléments.

De même, l'alphabet de sortie O_M est composé de tous les mintermes sur V_O et $|O_M| = 2^{|V_O|}$.

A partir de l'exemple de modèle de spécification présenté en Figure 1.10, l'ensemble des variables de sortie est $V_O = \{O_1, O_2\}$. L'alphabet de sortie de la machine de Mealy est alors $O_M = \{\overline{O_1}.\overline{O_2}, O_1.\overline{O_2}, \overline{O_1}.O_2, O_1.O_2\}$ et contient bien $2^2 = 4$ éléments.

La construction du modèle formel de la spécification sous la forme d'une machine de Mealy à partir d'un Grafcet sera développée dans le chapitre suivant.

1.3.2.3 Hypothèses

Plusieurs hypothèses sont retenues dans ces travaux concernant la machine de Mealy :

- La machine de Mealy est complète, c'est-à-dire que les fonctions de transition δ_M et de sortie λ_M sont définies pour toutes les combinaisons de variable d'entrée depuis tous les états.

$$\forall i_M \in I_M, \forall s_u \in S_M, \exists s_d \in S_M \text{ tel que } \delta_M(i_M, s_u) = s_d \tag{1.1}$$

$$\forall i_M \in I_M, \forall s_u \in S_M, \exists o_M \in O_M \text{ tel que } \lambda_M(i_M, s_u) = o_M \tag{1.2}$$

- La machine de Mealy est déterministe, c'est-à-dire que depuis chaque état de la machine de Mealy, il existe une seule valeur dans les fonctions de transition et de sortie pour chaque couple combinaison de variables d'entrée.

$$\begin{gathered}\forall (i_1, i_2) \in I_M^2, \forall (s_u, s_1, s_2) \in S_M^3, \text{ tels que } \delta_M(i_1, s_u) = s_1 \text{ et } \delta_M(i_2, s_u) = s_2 \\ s_1 \neq s_2 \implies i_1 \neq i_2\end{gathered} \tag{1.3}$$

$$\forall(i_1, i_2) \in I_M^2, \forall s_u \in S_M, \forall(o_1, o_2) \in O_M^2, \text{ tels que } \lambda_M(i_1, s_u) = o_1 \text{ et } \lambda_M(i_2, s_u) = o_2$$
$$o_1 \neq o_2 \implies i_1 \neq i_2 \tag{1.4}$$

- Une des conséquences des deux premières hypothèses est que la machine de Mealy est minimale, c'est-à-dire qu'il n'existe pas de machine de Mealy contenant moins d'état ou de transitions décrivant le même comportement (i.e. qui ait le même langage accepté).

- La machine de Mealy est fortement connexe, c'est-à-dire qu'entre chaque couple d'états, il existe au moins un chemin aller et retour composés d'une succession de transitions.

$$\forall(s_1, s_2) \in S_M^2, \exists(i_1, ..., i_j) \in I_M^j, \exists(o_1, ..., o_j) \in O_M^j \text{ tels qu'il existe un chemin}$$
$$s_1 \rightarrow^{i_1/o_1} s_i \rightarrow^{i_2/o_2} ... \rightarrow^{i_j/o_j} s_2 \tag{1.5}$$

- Chaque état de la machine de Mealy est distingable pas la valeur des variables de sortie. C'est-à-dire que toutes les transitions dont la destination est commune, et seulement ces transitions, sont étiquetées avec la même valeur des variables de sortie. Cette hypothèse est nécessaire pour rendre possible la détection d'un certain type de fautes qui sera détaillée par la suite.

La figure 1.11 présente une machine de Mealy qui vérifie les hypothèses retenues. Cette machine de Mealy correspond à un modèle de spécification avec deux variables d'entrée $V_I = \{a, b\}$ et une variable de sortie $V_O = \{o\}$.

L'alphabet d'entrée de cette machine est $I_M = \{\overline{a}.\overline{b}, a.\overline{b}, \overline{a}.b, a.b\}$, l'alphabet de sortie est $O_M = \{o, \overline{o}\}$, l'ensemble des états est $S_M = \{s_1, s_2\}$ avec s_1 l'état initial. La fonction de transition définit les 8 transitions ($\delta(s_1, \overline{a}.\overline{b}) = s_1$, $\delta(s_1, a.\overline{b}) = s_1$, $\delta(s_1, \overline{a}.b) = s_1$, $\delta(s_1, a.b) = s_2$, $\delta(s_2, \overline{a}.\overline{b}) = s_2$, $\delta(s_2, a.\overline{b}) = s_1$, $\delta(s_2, \overline{a}.b) = s_1$, $\delta(s_2, a.b) = s_2$) et la fonction de sortie définit les sorties émises pour chacune des transitions ($\lambda(s_1, \overline{a}.\overline{b}) = \overline{o}$, $\lambda(s_1, a.\overline{b}) = \overline{o}$, $\lambda(s_1, \overline{a}.b) = \overline{o}$, $\lambda(s_1, a.b) = o$, $\lambda(s_2, \overline{a}.\overline{b}) = o$, $\lambda(s_2, a.\overline{b}) = \overline{o}$, $\lambda(s_2, \overline{a}.b) = \overline{o}$, $\lambda(s_2, a.b) = o$).

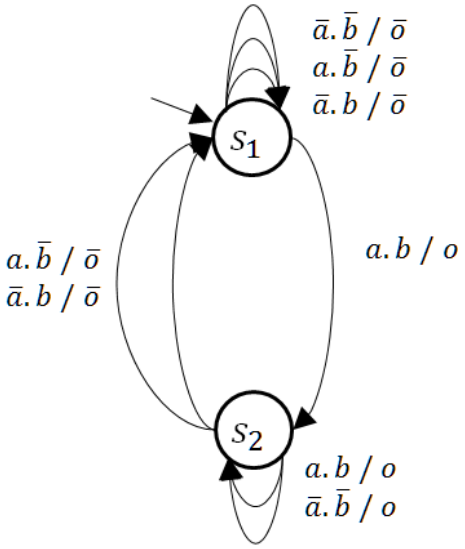


Figure 1.11 – Exemple simple de machine de Mealy

1.3.2.4 Fautes de transfert et de sortie

Le principe de la validation d'un contrôleur logique est de comparer le comportement du programme de commande implanté dans ce contrôleur et pouvant être représenté par une machine de Mealy M_{impl}, avec celui décrit par la spécification, elle aussi représentée par une machine de Mealy M_{spec}.

Lors de la comparaison de ces deux modèles, deux types de fautes peuvent apparaître :

- les fautes de sortie : l'état atteint est bien celui attendu mais la sortie émise n'est pas celle attendue,
- les fautes de transfert : la bonne sortie est émise mais le franchissement de la transition ne mène pas vers l'état attendu.

Si une faute est détectée sur l'une des transitions alors cette transition est déclarée non-valide. Si une transition non-valide est détectée dans le programme de commande, alors le contrôleur logique dans lequel il est implanté est lui aussi déclaré non-valide. La Figure 1.12 illustre la différence entre le modèle de comportement du programme de commande et le modèle de la spécification lorsque ces deux types de fautes sont présents.

Dans les travaux présentés dans ce mémoire de thèse, seule la détection de ces fautes est traitée. Le verdict de non-validité de l'implantation du programme de commande est rendu si une faute apparait, mais le type de faute n'est pas renseigné.

Comme les modèles de fautes prennent en compte les fautes de transfert, il est nécessaire de pouvoir identifier chacun des états du modèle de comportement du programme

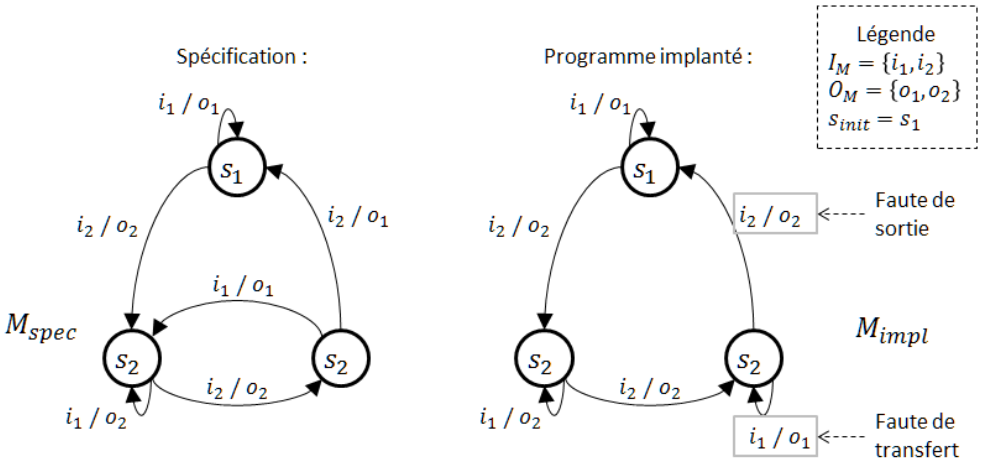


Figure 1.12 – Fautes de transfert et de sortie dans une machine de Mealy

de commande. Comme la seule information accessible est l'observation de la valeur des variables de sortie, l'hypothèse est formulée pour les modèles M_{spec} et M_{impl} que les états sont distingables par la valeur des variables de sorties émises en entrant dans cet état.

1.4 Brève présentation de quelques méthodes formelles d'analyse

De nombreuses techniques d'analyse formelle de contrôleurs logiques existent. Celles-ci peuvent être séparées en deux catégories distinctes : celles basées sur un modèle construit à priori du programme logique implanté dans le contrôleur et celles basées sur l'exécution de ce programme dans le contrôleur.

Dans un premier temps, la présentation des principales méthodes d'analyse pour la validation de contrôleurs logiques basées sur un modèle du programme implanté est proposée.

1.4.1 Méthodes opérant sur un modèle du programme de commande

Une première technique utilisant un modèle du programme de commande largement utilisée est la technique du Model-Checking (McMillan (1993), Clarke et al. (1999), Berard et al. (2001)). Cette technique permet de s'assurer qu'un modèle, qui peut être sous la forme d'un système de transition ou encore d'un réseau d'automates communiquant par exemple, vérifie un certain nombre de propriétés. Ces propriétés mathématiques peuvent être de diverses formes (logiques, séquentielles, temporisées ou non...) et

peuvent représenter des propriétés d'atteignabilité, de sûreté, de vivacité et d'équité. De plus, il est aussi possible de modéliser la partie opérative afin de la coupler au modèle du programme de commande pour se rapprocher de la configuration d'utilisation finale. Machado et al. (2006) a notamment prouvé qu'un modèle du programme implanté dans le contrôleur seul est préférable pour rendre un verdict sur des propriétés de sécurité alors qu'un modèle de ce programme couplé au modèle de sa partie opérative est préférable pour rendre un verdict sur des propriétés d'atteignabilité.

Une seconde technique utilisant un modèle du programme de commande est la technique de simulation (Bank (2010)). Cette technique repose sur l'étude du parcours de l'espace d'états d'un modèle en suivant un scénario défini par une séquence d'occurrences d'évènements d'entrée. La simulation permet de vérifier si les propriétés dans le cahier des charges sont vérifiées lors de la simulation du scénario. De la même manière, il est possible d'associer un modèle de la partie opérative au modèle du programme de commande pour la réalisation des scénarios de simulation.

Cependant, ces deux techniques sont basées sur la construction d'un modèle du programme de commande et non pas sur le programme implanté dans un contrôleur logique. Comme l'ambition des travaux présentés dans ce mémoire de thèse est principalement de fournir un verdict sur le programme de commande qui soit dans une configuration la plus proche possible de la configuration réelle d'utilisation, les techniques basées sur la validation de contrôleur logique exécutant le programme de commande seront privilégiées.

La section suivante présente donc deux techniques de validation de contrôleurs logiques que sont le test de conformité et la simulation Hardware-In-the-Loop, ainsi qu'une technique d'analyse formelle qu'est l'identification.

1.4.2 Méthodes opérant sur un API exécutant un programme de commande

1.4.2.1 Test de conformité

Pour qu'un système logique soit annoncé valide, il faut être capable d'analyser et de fournir un verdict pour toutes les évolutions considérées durant la phase de validation. Une technique capable de garantir cela est le test de conformité.

Cette technique repose sur un principe illustré en Figure 1.13. Pour cela, l'API qui

exécute le programme de commande est **isolé de la partie opérative**. L'API est considéré comme une boite noire, c'est-à-dire qu'aucune information n'est à priori connue. Seule la connaissance d'un modèle formel de la spécification, qui peut prendre de nombreuses formes, suffit à réaliser la validation du programme de commande implanté dans l'API (Chow (1978), Lee and Yannakakis (1996), Zhu and He (2002), Brinksma and Tretmans (2001), Provost et al. (2011a)).

En effet, la spécification décrit quelles sont les sorties émises en fonction des entrées perçues par l'API. Il est donc possible de savoir, lorsque l'on applique une valeur d'entrée aux bornes de l'API, quelle est la valeur de sortie que celui-ci doit émettre pour que son comportement soit déclaré valide. Il suffit alors d'observer les sorties qu'il émet effectivement pour déclarer ou non cette validité. Lorsque le verdict de validité est déclaré, on dit que l'API est conforme à sa spécification.

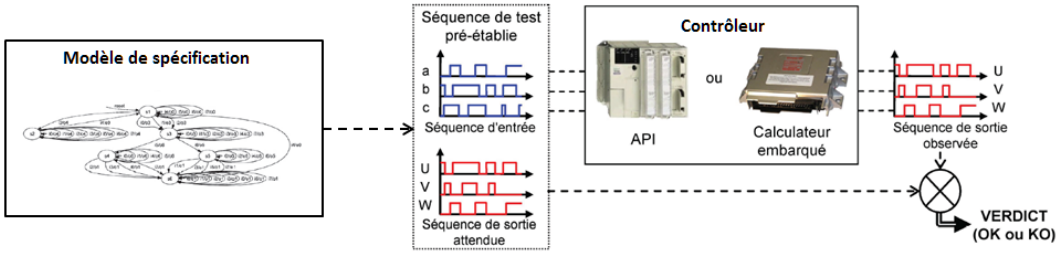


Figure 1.13 – Principe du test de conformité

Plus précisément, un test de conformité est donc composé de deux étapes distinctes : **la construction d'une séquence de test** et **l'exécution de cette séquence de test**.

La construction de la séquence se fait à partir de la connaissance du modèle de spécification et prend la forme d'une séquence de combinaisons de valeurs d'entrée. Cette construction nécessite trois autres éléments :

- Une technique de parcours de chemin, comme par exemple celles développées dans Naito and Tsunoyama (1981), Lee and Yannakakis (1996), Dorofeeva et al. (2010) ou Li and Kumar (2012),

- un objectif de test, qui sert à définir la complétude de la séquence de test. Lorsque le modèle formel de la spécification est une machine de Mealy, un objectif de test fréquent est que toutes les transitions de la machine de Mealy aient été franchies au moins une fois.

- Un ou plusieurs critères d'optimisation de la séquence de test, comme par exemple qu'elle soit de longueur minimale.

L'exécution de la séquence de test consiste en l'application de la séquence de test précédemment construite aux bornes du contrôleur, l'observation des sorties émises en réponse et l'analyse des sorties observées. Cette analyse est réalisée à l'aide d'une relation de conformité qui définit la relation que doit vérifier la séquence de valeurs de sortie observée pour être considérée comme conforme. Elle est généralement définie comme une égalité entre les valeurs de sortie attendues et observées comme Tretmans (1996) pour les systèmes de transition ou Ponce de Leon et al. (2012) pour les systèmes concurrents par exemple.

Il est ainsi possible de garantir un verdict de conformité défini par la relation de conformité pour la totalité de l'espace d'état défini dans l'objectif de test. Deux termes doivent alors être définis afin de caractériser les situations où le verdict du test (ou de toute autre technique de validation) rendu n'est pas correct. Soit l'implantation est déclarée conforme alors que des non-conformités existent et le verdict est alors dit *erroné*, soit l'implantation est déclarée non-conforme à tort et le verdict est dit *biaisé*.

Avec une technique de test de conformité, le contrôleur demeure cependant isolé de la partie opérative tout au long de l'exécution du test. Cela signifie que le contrôleur logique n'est pas dans sa configuration d'utilisation réelle. Une seconde méthode de validation, appelée simulation Hardware-In-the-Loop (HIL), propose une solution pour valider le programme de commande tout en prenant en compte la partie opérative.

1.4.2.2 Simulation HIL

La simulation HIL est une technique répondant à une volonté combinée de valider un programme de commande implanté dans un API tout en considérant son utilisation couplée à une partie opérative. Cependant, dans des phases de conception telle que le prototypage, il est souvent dangereux ou impossible de coupler le contrôleur logique à une partie opérative réelle. Cela peut être dû au fait que celle-ci n'a pas encore été réalisée ou qu'une non-validité du programme de commande pourrait entrainer des conséquences trop importantes sur la partie opérative.

Une alternative est donc de construire un modèle de cette partie opérative qui simulera son fonctionnement. Des logiciels comme ControlBuild (http ://www.3ds.com/products-

services/catia/capabilities/systems-engineering/embedded-systems/controlbuild/) ou Modelica (https ://www.modelica.org/) proposent des interfaces de construction de modèles de partie opérative ainsi que des interfaces de communication qui permettent d'envoyer et de recevoir les valeurs d'entrée/sortie depuis l'extérieur. Il est donc possible de connecter le contrôleur logique à ce type de logiciel.

Une fois le contrôleur logique connecté à une partie opérative simulée, les techniques de simulation HIL (Isermann et al. (1999)), comme les techniques de simulation classiques, consistent en la définition et l'exécution de scénarios. Ces scénarios mettent en scène des situations précises et visent à valider la réponse de l'API à des évolutions de comportement de la partie opérative. La figure 1.14 illustre cette méthode.

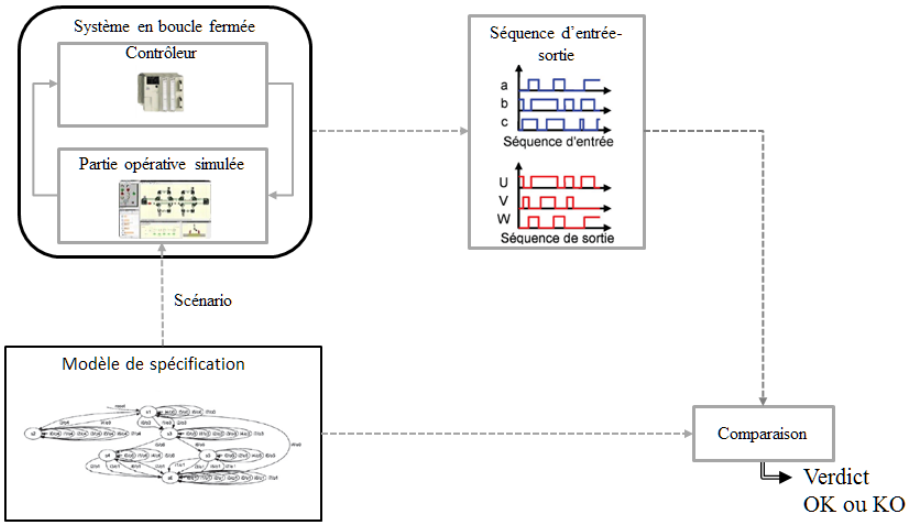


Figure 1.14 – Principe de la simulation HIL

Cette méthode de validation est très présente dans les milieux industriels. On peut notamment la retrouver pour répondre à des problématiques de transport (Bullock et al. (2004), Schlager (2008)...), d'énergie (Lu et al. (2007)) ou de défense (Bacic (2005)).

Cependant, comme elle nécessite la définition de scénarios basés sur les réactions possibles de la partie opérative, ce type de méthode de validation est plus adapté à des besoins de validation pour des situations particulières et critiques que pour une validation du comportement en général de la logique de contrôle.

1.4.2.3 Identification

Les techniques d'identification de système à évènements discrets ne sont pas des techniques de validation mais des techniques d'analyse formelle de systèmes logiques.

Elles consistent en la construction d'un modèle formel à partir de l'observation du comportement d'un système logique. La Figure 1.15 illustre le principe de cette technique. Comme ces techniques ne nécessitent pas de connaissance à priori du modèle du système logique, elles sont principalement utilisées lorsque la connaissance précise du modèle du comportement réel est requise. C'est notamment le cas lorsque l'objectif est du diagnostic ou du reverse-engineering. Estrada-Vargas et al. (2010) présente les principales applications et techniques pour l'identification.

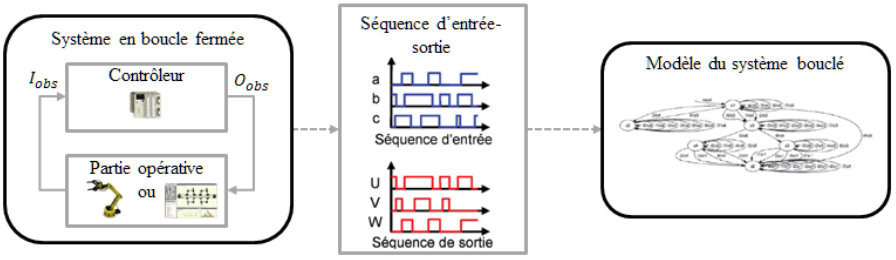


Figure 1.15 – Principe de l'identification

Les techniques d'identification pour les systèmes à évènements discrets peuvent se scinder en deux catégories distinctes :

- Les techniques d'identification en boite noire, où le modèle à identifier est inconnu. Seules les informations sur les entrées lues et les sorties émises, si elles existent, sont connues.

- Les techniques d'identification en boite grise, où certaines caractéristiques du modèle à identifier sont connues. Cela peut être une bornes maximale ou le nombre d'états, de transitions, la structure de l'automate, ...

Les techniques d'identification en boite noire sont généralement appliquées à des systèmes à évènements discrets de type réseau de Petri (Meda et al. (1998), Giua and Seatzu (2005), Estrada-Vargas et al. (2013)) ou de type automates à états finis (Klein et al. (2005), Roth et al. (2010)). Les techniques d'identification en boite grise, elles, ont principalement été développées pour des réseau de Petri en supposant une borne maximale du nombre de places connues (Dotoli et al. (2008)) ou le nombre exact de places et de transitions connues (Cabasino et al. (2007)).

Ces techniques ont pour avantage de se baser sur l'observation des évolutions d'un système physique afin de déterminer un modèle de comportement et se passent donc de

la connaissance à priori de celui-ci. Cela sera utile dans la suite de ce mémoire, dans le cadre du développement de nouvelles techniques de validation, lorsqu'il sera question de construire un modèle formel du comportement du système composé du contrôleur à valider couplé sa partie opérative.

1.5 Objectif des travaux

L'objectif de ces travaux est de développer des techniques de validation de contrôleurs logiques basées sur l'exécution par un API d'un programme de commande et non pas sur un modèle du comportement de ce programme de commande. La réalisation de cet objectif est découpée en trois parties distinctes :

Dans un premier temps, le passage d'une spécification issue du cahier des charges sous la forme d'un Grafcet à un modèle du comportement du programme de commande conforme à la spécification sous la forme d'une machine de Mealy. Cette étape est nécessaire afin de pouvoir comparer ensuite le comportement du contrôleur logique avec le comportement attendu par la spécification.

Ensuite, deux approches distinctes seront étudiées pour la validation du contrôleur logique. Une première basée sur des techniques de test de conformité et une seconde basée sur des techniques de validation en boucle fermée.

1.5.1 Du modèle de spécification Grafcet au modèle formel

Des travaux existent déjà traitant de la construction de modèle formel d'une logique de commande à partir de spécification Grafcet et sont présentés dans Provost et al. (2011b). Cette construction répond à des règles précises sur les évolutions dans un Grafcet ainsi que sur les algorithmes d'interprétation en accord avec la norme IEC 60848 (2002). Cependant, les pratiques industrielles font apparaitre des différences avec les recommandations de la norme et ne sont actuellement pas compatibles avec la méthode précédemment développée. Un premier apport de ce mémoire de thèse est donc d'étendre les travaux présentés dans Provost et al. (2011b) afin de les adapter aux pratiques industrielles.

1.5.2 Contribution au test de conformité

Les travaux présentés dans Provost et al. (2011a) proposent une technique adaptée au test de conformité de contrôleurs logiques. Les résultats obtenus pour les phases de construction et de l'exécution de la séquence de test sont novateurs mais certaines lacunes persistent :

- L'étape de comparaison des séquences de sortie observées avec les séquences de sortie attendues est décrite en langage naturel comme une égalité entre les sorties émises et celles attendues mais aucune relation de conformité n'est formellement définie.
- Un phénomène de lecture asynchrone d'évènements synchrones a été détecté lors de l'exécution de la séquence de test. La solution proposée dans Provost et al. (2014) qui vise à supprimer de la séquence les évènements synchrones ne permet pas de maintenir la garantie de l'objectif de test.

Une extension des travaux précédents est donc proposée dans ce mémoire afin de combler ces lacunes.

1.5.3 Contribution à la validation de système bouclé

Le test de conformité isole le contrôleur logique de sa partie opérative et le place donc dans une configuration qui diffère de celle dans lequel il sera lors de son utilisation. Comme l'étape de validation sert à déterminer si l'on fabrique le bon produit, il est judicieux de placer celui-ci dans une situation qui est celle de son utilisation. Dans le cas de validation de contrôleur logique, cela signifie qu'il soit connecté à sa partie opérative.

Une nouvelle technique de validation est donc proposée qui conserve le contrôleur logique connecté à sa partie opérative. Cette technique, appelée validation en boucle fermée, est basée sur l'observation des évolutions de l'ensemble couplé contrôleur - partie opérative et sur la comparaison des évolutions observées avec le modèle formel de la spécification.

Cette nouvelle technique requiert la définition de deux éléments afin de la rendre opérationnelle :

- Un critère qui définisse à quel moment le contrôleur logique est déclaré valide, similaire à la notion d'objectif de test pour le test de conformité.

- Une relation qui juge de la validité ou non des entrées-sorties observées en fonction du modèle de spécification, similaire à la relation de conformité.

Les phases d'observation, de construction des modèles et d'analyse des séquences observées seront détaillées.

Chapitre 2

Construction du modèle formel de spécification sous forme d'une machine de Mealy

Sommaire

Introduction

Lorsqu'un modèle de spécification sous la forme d'un Grafcet est implanté dans un contrôleur logique, deux opérations sont nécessaires :

- Il faut choisir le langage de programmation (C, Ladder, SFC, ...) dans lequel sera rédigé le programme de commande,
- Il faut choisir une interprétation du Grafcet.

En effet, si le modèle Grafcet décrit le comportement attendu d'un système logique, il lui manque la sémantique opérationnelle qui définit comment le programme de commande va gérer les entrées et sorties. Cette sémantique opérationnelle est décrite dans des algorithmes d'interprétation.

Dans le cas du Grafcet, deux interprétations sont possibles. Une première, dite avec recherche de stabilité, qui est normalisée et une seconde, dite sans recherche de stabilité, qui est la plus communément utilisée pas les industriels. En effet, le premier type d'interprétation limite les émissions courtes (d'une durée d'un temps de cycle API) des variables de sortie. Cependant, les opérations nécessaires à ce filtrage des sorties émises induit une augmentation du temps de calcul avant émission des sorties suite à un changement de valeur des variables d'entrée. Cela peut se révéler incompatible avec les exigences de réactivité des procédés industriels. C'est pourquoi la seconde interprétation, sans limitation des sorties émises et qui présente de meilleures performances en terme de réactivité, est utilisée pour les applications industrielles.

Ce chapitre s'intéresse donc à la construction du modèle formel de la spécification qui décrit le comportement d'un programme de commande conforme à cette spécification sous la forme d'une machine de Mealy. Une technique ayant déjà été développée pour l'interprétation normalisée, ce chapitre propose un algorithme de construction de modèle formel pour la seconde interprétation ainsi que la mise en évidence des différences entre les deux modèles formels produits. Ces travaux ont fait l'objet d'une communication en conférence nationale Guignard and Faure (2013a) ainsi qu'en revue nationale Guignard and Faure (2013c).

Ce chapitre se découpe en quatre parties distinctes :

- Tout d'abord, un rappel sur les deux types d'interprétation du Grafcet ainsi que les

cas où ils influent sur les variables de sortie émises par le programme de commande implanté dans l'API est proposé.

- Ensuite, la seconde section présente un rappel synthétique d'une méthode déjà existante pour construire un modèle de spécification sous la forme d'une machine de Mealy pour l'interprétation préconisée dans la norme.

- La troisième section propose, elle, un algorithme de construction du modèle de spécification sous la forme d'une machine de Mealy pour la seconde interprétation plus répandue dans le milieu industriel.

- Une comparaison des deux modèles de spécification obtenus en fonction de l'interprétation du Grafcet est présentée dans la quatrième section.

- Enfin, une synthèse est proposée dans la cinquième section.

2.1 Interprétation du Grafcet avec ou sans recherche de stabilité

Le Grafcet est un langage de spécification décrivant un comportement théorique mais qui doit être complété par une interprétation servant à définir la manière dont le programme de commande implanté dans le contrôleur va calculer les divers changements de variables logiques ou d'état. Dans certains cas, l'interprétation choisie peut influer la manière dont seront traitées certaines évolutions par le programme de commande.

C'est notamment le cas lorsque deux transitions successives du Grafcet sont franchies sans changement de variables d'entrée, appelé évolution fugace (IEC 60848 (2002), page 13). Une évolution fugace se définit par le franchissement successif de transitions ou d'ensembles de transitions sans changement de valeur des variables d'entrée. Cela signifie que, après une premier franchissement de transition(s), il existe au moins une transition du Grafcet qui soit validée et dont la réceptivité est vraie. Selon la règle n°2 des évolutions du Grafcet, ce nouvel ensemble de transitions doit donc être immédiatement franchi, et ainsi de suite jusqu'à ce qu'aucune transition ne soit plus franchissable.

La Figure 2.1 propose un exemple simple où la combinaison de valeurs des variables d'entrée entraîne ou non une évolution fugace du Grafcet : les transitions $t1$ et $t2$ sont

toutes deux franchissables lorsque les variables a et b sont à $Vrai$, le Grafcet étant dans sa situation initiale.

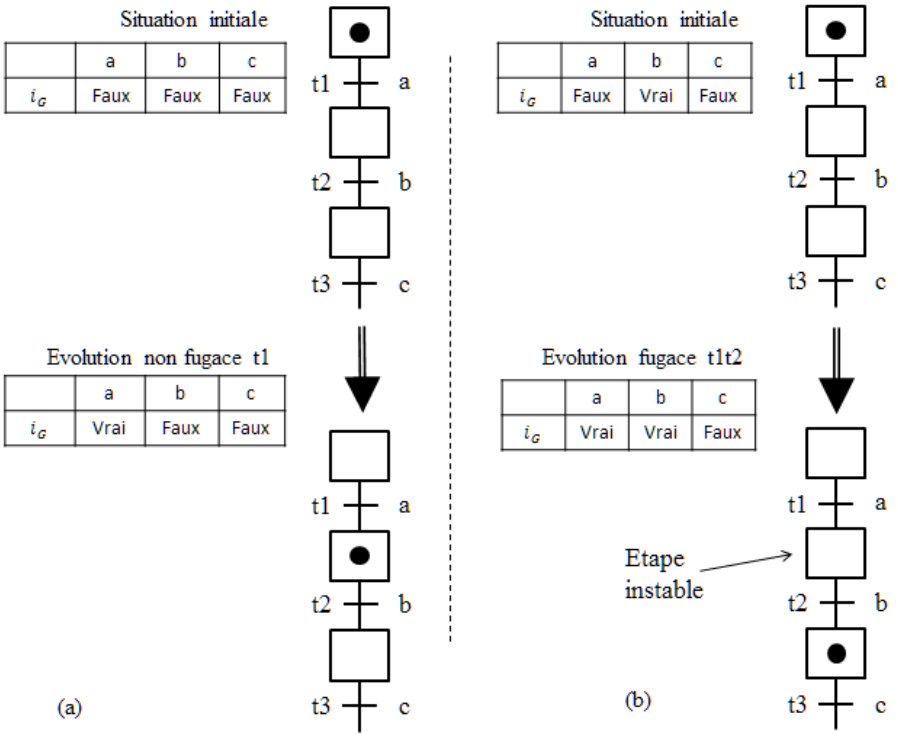


Figure 2.1 – Evolution non fugace (a) ou fugace (b) d'un Grafcet

Une définition formelle d'une évolution fugace impliquant deux franchissements successifs de transition est proposée ci-dessous. Il est possible de généraliser cette définition à n franchissements successifs en vérifiant la seconde condition après franchissement des transitions jusqu'à ne plus trouver de transitions franchissables.

Definition 2.1. *Evolution fugace*

Soit un Grafcet G dans une situation (ensemble d'étapes actives) $S_{Act} \subset S_G$.

Soit $T_{valid} \subset T_G$ l'ensemble des transitions validées, c'est-à-dire l'ensemble des transitions dont toutes les étapes amont sont actives : $T_{valid} = \{t \in T_G | t \centerdot {S_U}^3 \in S_{Act}\}$.

Si il existe une combinaison v_I de valeur des variables d'entrée telle que :

- *Il existe $t_1 \in T_{valid}$ telle que $t_1 \centerdot FC(v_I, X_G) = True$*
- *Soit le nouvel ensemble d'étapes actives après le franchissement de la transition t_1, $S_{New} = (S_{Act}/t_1 \centerdot S_U) \cup t_1 \centerdot S_D$, il existe $t_2 \in T_G, t_2 \notin T_{valid}$ telle que $t_2 \centerdot S_U \in S_{New}$ et $t_2 \centerdot FC(v_I, X_G) = True$*

Alors, l'évolution t_1t_2 est dite fugace pour la valeur des variables d'entrée v_I et l'ensemble $t_1 \centerdot S_D \cap t_2 \centerdot S_U$ est l'ensemble des étapes instables.

Un Grafcet est dit dans une situation stable pour une combinaison de valeurs des variables d'entrée v_I si aucune transition n'est franchissable pour cette combinaison :

Definition 2.2. *Situation stable*
Soit un Grafcet G comportant un ensemble d'étapes actives à l'instant considéré $S_{Act} \in S_G$.
Soit $T_{valid} \subset T_G$ l'ensemble de transitions validées $T_{valid} = \{t \in T_G | t \centerdot S_U \in S_{Act}\}$.
Soit v_I une combinaison de valeurs des variables d'entrée.
Si $\forall t \in T_{valid}$, $t \centerdot FC(v_I, X_G) = False$, alors le Grafcet est dit dans une situation stable (i.e. seul un changement de valeur des variables d'entrée pourra déclencher le franchissement d'au moins une transition).

Un Grafcet contient un cycle d'instabilité s'il existe une évolution fugace telle qu'un état stable ne soit jamais atteint. On suppose dans ces travaux que **le modèle de spécification ne contient aucun cycle d'instabilité**.

L'exécution d'une évolution fugace peut être interprétée de deux manières différentes :

- Interprétation avec recherche de stabilité : on considère que l'activation et la désactivation des étapes intermédiaires est instantanée. C'est-à-dire que sur changement de valeur des variables d'entrée, la situation stable, si elle existe, est instantanément atteinte et seules les actions mémorisées associées aux étapes intermédiaires sont prises en compte (algorithme 1).

- Interprétation sans recherche de stabilité : on considère que l'activation et la désactivation des étapes intermédiaires n'est pas instantanée. C'est-à-dire que toutes les étapes intermédiaires d'une évolution fugace sont activées pendant un temps non nul et que les actions continues associées à ces étapes sont aussi émises pendant ce temps d'activation (algorithme 2).

3. $a \centerdot b$ signifie l'élément b de a.

Algorithme 1 Inteprétation du Grafcet avec recherche de stabilité

```
1: Stabilité := Faux ;
2: Déterminer l'ensemble des transitions franchissables ;
3: while Stabilité = Faux do
4:     Déterminer l'ensemble des nouvelles étapes actives ;
5:     Mettre à jour les actions mémorisées ;
6:     Déterminer l'ensemble des transitions franchissables ;
7:     if l'ensemble des transitions franchissables est vide then
8:         Stabilité := Vrai ;
9:     end if
10: end while
11: Calculer la valeur des sorties contrôlées par des actions continues associées aux étapes
    actives ;
```

Algorithme 2 Inteprétation du Grafcet sans recherche de stabilité

```
1: Déterminer l'ensemble des transitions franchissables ;
2: Déterminer l'ensemble des nouvelles étapes actives ;
3: Mettre à jour les actions mémorisées ;
4: Mettre à jour les actions continues ;
```

La comparaison de ces deux algorithmes met bien en évidence la différence en terme de réactivité des deux interprétations. En effet, pour une évolution fugace comportant n franchissements successifs de transition, il faut dans le cas d'une interprétation avec recherche de stabilité calculer n fois les transitions franchissables, les nouvelles étapes actives et les sorties associées aux actions mémorisées (boucle *While* des lignes 2 à 10) avant de mettre à jour l'ensemble des sorties. Alors qu'avec une interprétation sans recherche de stabilité, les sorties sont mises à jours après chaque franchissement de transition. Dès 3 franchissements successifs de transitions, le temps de calcul est plus que doublé dans le cas d'une interprétation avec recherche de stabilité par rapport à une interprétation sans recherche de stabilité.

Le grafcet doit donc être traduit en un modèle qui soit capable de prendre en compte l'interprétation retenue pour sa traduction en un code logique à implanter dans le contrôleur. La section suivante rappelle les résultats présentés dans Provost et al. (2011b) à propos de la construction du modèle formel de spécification pour une interprétation avec recherche de stabilité.

2.2 Rappels : Construction du modèle formel de spécification pour une interprétation avec recherche de stabilité

Afin de construire une machine de Mealy à partir d'un Grafcet, Provost et al. (2011b) propose de considérer l'interprétation préconisé par la norme IEC 60848 (2002) : l'exécution avec recherche de stabilité. Pour cela, il passe par un modèle intermédiaire sous la forme d'un automate à état fini, appelé automate des localités stables, qui suppose une interprétation avec recherche de stabilité. Cette section présente la définition d'un ALS et le principe de la construction de cet ALS ainsi que de la machine de Mealy à partir de la spécification Grafcet.

2.2.1 Du Grafcet à l'automate des localités stables (ALS)

Un automate des localités stables est formellement défini comme un $5-uplet$ $(I_{AL}, O_{AL}, L, l_{Init}, Evol)$ où :

- $I_{AL} = V_I$ est l'ensemble des entrées logiques,
- $O_{AL} = V_O$ est l'ensemble des sorties logiques,
- L est l'ensemble des localités,
- l_{Init} est la localité initiale,
- $Evol$ est l'ensemble des évolutions de l'automate.

Chaque **localité** est définie par l'ensemble des étapes actives du Grafcet, l'ensemble des sorties émises ainsi que par une condition booléenne sur les variables d'entrée représentant la condition de non-évolution de cette localité, dite condition de stabilité. Une localité est donc formellement définie par $l \in L$, un $3-uplet$ $(S_{Act}, O_{Em}, C(V_I))$ où :

- S_{Act} est la situation du Grafcet G,
- O_{Em} est un sous-ensemble des sorties de G définissant les sorties émises dans cette situation,
- $C(V_I)$ est la condition de stabilité de la localité, sous la forme d'une expression booléenne sur V_I correspondant à la condition pour qu'aucune transition de G

validée pour la situation courante ne soit franchissable.

Une **évolution** d'un ALS correspond à une séquence de franchissements de transitions entre deux situations stables pour le Grafcet et est représentée par un arc orienté entre deux localités de l'ALS. Chaque évolution $evol \in Evol$ est donc définie par un $3-uplet$ $(l_U, l_D, E(V_I))$ où :

- $l_U \in L$ est la localité amont de l'évolution,
- $l_D \in L$ est la localité aval de l'évolution,
- $E(V_I)$ est la condition de franchissement de l'évolution, sous la forme d'une expression booléenne sur V_I correspondant à la condition sur V_I pour le franchissement d'une séquence de transitions de G.

La Figure 2.2 illustre les informations portées par les localités ainsi que par les évolutions.

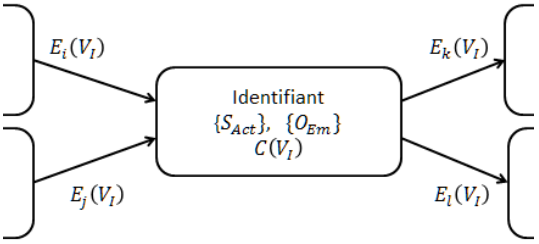


Figure 2.2 – Localité d'un ALS

La génération de l'ALS se fait en parcourant l'ensemble de son espace d'état. Depuis la localité initiale puis pour chaque localité créée, l'algorithme recherche puis crée l'ensemble des évolutions. Chaque évolution peut correspondre à :

- un franchissement d'une séquence d'ensembles de transitions simultanément franchissables. Cette évolution conduit à une localité représentant une situation du Grafcet différente de la situation avant franchissement.
- Un changement de valeur des actions sans franchissement de transition du Grafcet. Cela peut se produire dans le cas où des actions conditionnelles sont associées à des

étapes du Grafcet. La localité de destination représentera alors la même situation du Grafcet, mais avec une valeur des variables de sortie différente.

Pour chacune de ces évolutions, la localité aval est créée si elle est atteinte pour la première fois et sera ensuite étudiée. Les détails de cette opération sont formellement présentés dans Provost et al. (2011b).

La Figure 2.3 présente l'automate des localités stables obtenu à partir de la spécification Grafcet présentée en Figure 1.10.

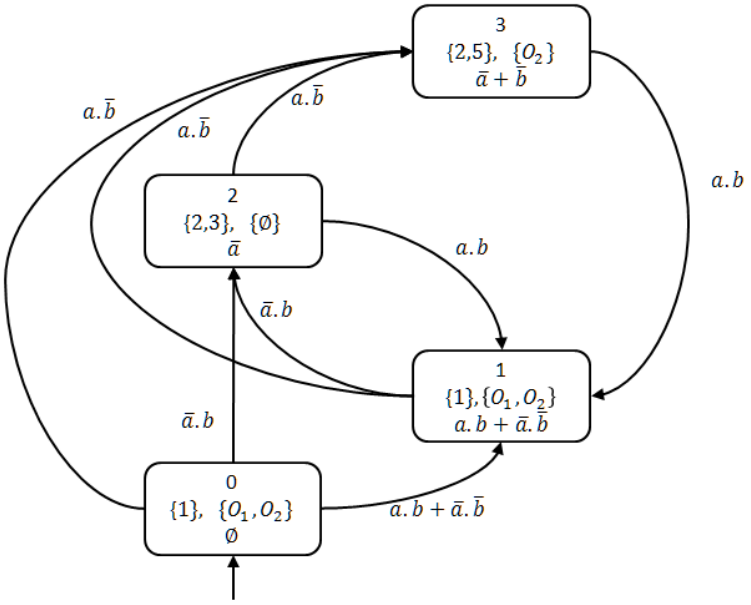


Figure 2.3 – ALS déduit du Grafcet présenté Figure 1.10

On peut remarquer la forme particulière de la localité initiale dont la condition de stabilité est nulle et qui ne présente que des évolutions dont elle est la localité amont. Cette particularité est nécessaire afin de garantir que lors de l'initialisation du système logique, le modèle formel va instantanément atteindre la situation stable associée aux valeurs des variables d'entrée perçues. Ainsi, lors de l'initialisation, l'automate va atteindre un état stable correspondant à l'absence (localité 1) ou la présence (localités 2 et 3) de franchissement de transitions ou de changement de valeur des variables de sortie dues aux valeurs des variables d'entrée.

2.2.2 De l'ALS à la machine de Mealy

L'ALS définit précédemment est un modèle intermédiaire permettant d'obtenir une machine de Mealy à partir d'une spécification Grafcet interprétée avec recherche de

stabilité.

La structure de la machine de Melay est déduite de celle de l'ALS en effectuant les transformations suivantes :

- Un état dans la machine de Mealy est créé et associé à chaque localité sauf la localité initiale de l'ALS.

- La localité initiale n'apparait plus dans la machine de Mealy qui définit son état initial en fonction des valeurs des variables d'entrée. Celle-ci se place en effet directement dans l'état stable associée à la valeur initiale des variables d'entrée.

- Les transitions d'une machine de Mealy sont étiquetées avec des combinaisons de valeur de variables d'entrée et non pas des expressions booléennes. Chaque expression booléenne étiquetant une évolution entre deux localités est décomposée en une somme de mintermes. Chaque minterme est ensuite associé en tant qu'étiquette à une transition de la machine de Mealy reliant les deux états construits à partir des localités amont et aval.

- Les informations concernant la valeur des variables de sortie sont associées aux transitions et non plus aux localités. En effet, à chaque transition de la machine de Mealy est associé un élément de l'alphabet de sortie. La valeur de cet élément, défini comme un minterme sur les variables de sortie, est calculée à partir des sorties émises dans la localité associée à l'état aval de la transition considérée.

- La condition de stabilité associée à chaque localité se traduit sous forme de self-loops[4] sur l'état associé machine de Mealy. Cette condition de stabilité, sous la forme d'une expression booléenne sur les variables d'entrée, est décomposée en une somme de mintermes. Chaque self-loop porte en élément d'alphabet d'entrée un des mintermes de la condition de stabilié et en élément d'alphabet de sortie le minterme sur les variables de sortie correspondant aux sorties émises pour la localité considérée. Il faut donc créer autant de self-loops que de mintermes composant la condition de stabilité et toutes ces self-loop porteront le même élément de l'alphabet de sortie.

4. transition ayant le même état amont et aval

Une présentation formelle du passage d'un ALS à une machine de Mealy est proposée dans Provost et al. (2011b).

La Figure 2.4 présente la machine de Mealy obtenue à partir de la spécification Grafcet présentée en Figure 1.10 interprétée avec recherche de stabilité.

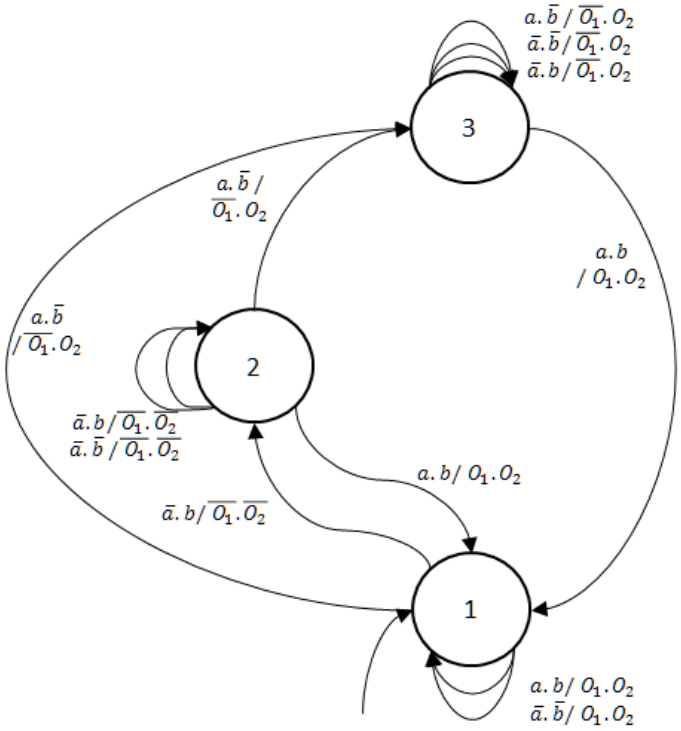


Figure 2.4 – Machine de Mealy correspondant à une interprétation avec recherche de stabilité issue de l'AL en Figure 2.3

2.3 Contribution à la construction du modèle formel de spécification pour une interprétation sans recherche de stabilité

2.3.1 Algorithme d'exécution du Grafcet sans recherche de stabilité

L'interprétation avec recherche de stabilité est le mode d'interprétation qui a été retenu dans Provost et al. (2011b) pour la construction de l'ALS à partir d'un Grafcet. Cependant, bien que cette interprétation soit celle préconisée par la norme, la présence de la boucle de recherche de stabilité peut nécessiter des temps de calcul conséquents qui sont incompatibles avec les exigences de réactivité des procédés industriels. C'est pourquoi nombre d'entre eux utilisent le second type d'interprétation qui est sans recherche de stabilité, présenté dans l'algorithme 2. La suite de cette section propose la définition et la méthode de construction d'un automate des localités (AL) qui décrit l'exécution

de la spécification Grafcet selon une interprétation sans recherche de stabilité. Une fois ce nouveau modèle intermédiaire construit, la technique permettant le passage vers la machine de Mealy restera inchangée.

2.3.2 Définition d'un automate des localités

Un AL possède la même structure qu'un ALS. Il est donc défini comme un $5-uplet$ $(I_{AL}, O_{AL}, L, l_{Init}, Evol)$ où :

- $I_{AL} = V_I$ est l'ensemble des entrées logiques,
- $O_{AL} = V_O$ est l'ensemble des sorties logiques,
- L est l'ensemble des localités tel que chaque élément $l = (S_{Act}, O_{Em}, C(V_I))$,
- l_{Init} est la localité initiale,
- $Evol$ est l'ensemble des évolutions de l'automate des localités tel que chaque élément $evol = (l_U, l_D, E(V_I))$.

Deux propriétés de cet automate, aussi valables pour l'interprétation avec recherche de stabilité, se traduisent par des expressions sur les conditions de franchissement et de stabilité qu'il doit vérifier :

- Le déterminisme.

$$\begin{gathered} \forall l \in L, \forall\ evol_i = (l_{Ui}, l_{Di}, E_i(V_I)) \in Evol \\ \forall\ evol_j(l_{Uj}, l_{Dj}, E_j(V_I)) \in Evol,\ i \neq j \text{ où :} \\ evol_i \in \{Evol | l_{Ui} = l\} \text{ et } evol_j \in \{Evol | l_{Uj} = l\} \text{ tels que} \\ (E_i(V_I)) \wedge (E_j(V_I)) = Faux \end{gathered} \tag{2.1}$$

- La complétude.

$$\begin{gathered} \forall\ l = (S_{Act}, O_{Em}, C(V_I)) \in L, \\ \forall\ evol_k = (l_{Uk}, l_{Dk}, E_k(V_I)) \in Evol, \\ C(V_I) = \overline{\sum_{l_{Uk}=l} E_k(V_I)} \end{gathered} \tag{2.2}$$

2.3.3 Construction d'un automate des localités à partir d'un Grafcet

2.3.3.1 Rappels sur la définition des conditions de franchissement des transitions du Grafcet

A partir des règles n°2 et n°4 d'évolution du Grafcet rappelées en section 1.3.1, il est possible de définir les relations logiques exprimant les conditions pour le franchissement de transitions. Ces relations sont nécessaires pour la construction d'un automate des localités, stable ou non. Les règles n°1, n°3 et n°5 décrivent, elles, la situation initiale et les changements d'étapes actives. Elles ne peuvent être exprimées sous la forme de relations logiques mais sont utilisées au sein de l'algorithme de construction de l'automate qui est détaillé par l'Algorithme 3 donné en page 50.

La règle n°2 présente les conditions nécessaires au franchissement d'une transition d'un Grafcet : toutes les étapes directement en amont de la transition doivent être actives, ce qui signifie que la transition est validée, et la réceptivité associée à la transition doit être vérifiée. Ainsi, le pré-requis pour le franchissement d'une transition peut être mathématiquement établi en remplaçant les variables d'activité d'étape par leur valeur dans la situation considérée :

Soit S_{Act} la situation courante du Grafcet,

Soit une transition $t = (S_U, S_D, FC(V_I, X_G)) \in T_G$, l'expression de la condition de franchissement pour cette situation est :

$$F_t(V_I) = FC(V_I, X_G) \tag{2.3}$$

$$avec \begin{cases} \forall s \in S_{Act} & X_s = Vrai \\ \forall s \notin S_{Act} & X_s = Faux \end{cases}$$

De plus, une transition t est franchie pour une combinaison de valeurs de variables d'entrée donnée si elle est validée et que la réceptivité associée à cette transition est Vrai pour ces valeurs de variables d'entrée, i.e. :

Soit $t = (S_U, S_D, FC(V_I, X_G)) \in T_{valid}$, $\exists v_I \in V_I$ tels que :

$$F_t(v_I) = Vrai \tag{2.4}$$

Soit le Grafcet présenté en Figure 1.10 dans une situation où les étapes 2 et 3 sont actives. Les transitions $t2$ et $t3$ sont toutes deux validées. La réceptivité de la transition $t2$ est $a.b$; cela signifie qu'en prenant $v_I = (a = Vrai, b = Vrai)$, la condition $F_{t2} =$

$a.b = Vrai$ est vérifiée et la transition est franchie.

La règle n°4 spécifie que plusieurs transitions peuvent être franchies simultanément. La condition de franchissement d'un ensemble de transitions $F_\tau(V_I)$ doit donc être définie comme suit :

Soit $\tau \subseteq T_{valid}$ un sous-ensemble de transitions validées. La condition de franchissement de ce sous-ensemble de transitions est la conjonction des conditions de franchissement de chaque transition de ce sous-ensemble avec le complément des conditions de franchissement des transitions validées qui n'appartiennent pas à ce sous-ensemble.

$$F_\tau(V_I) = \prod_{\substack{t \in T_{valid} \\ t \in \tau}} F_t(V_I) \wedge \prod_{\substack{t \in T_{valid} \\ t \notin \tau}} \overline{F_t(V_I)} \tag{2.5}$$

Cependant, cette relation n'est pas vérifiée pour tous les sous-ensembles de transitions validées τ. En effet, il n'est pas toujours possible de trouver une combinaison de valeurs des variables d'entrée v_I qui satisfasse cette relation. Un sous-ensemble de transitions simultanément franchissables SFT, pour une situation du Grafcet S_{Act} donnée, est un sous-ensemble de transitions validées pour lequel il existe au moins une valeur de $v_I \in V_I$ telle que la relation (2.5) soit vraie.

$$\exists v_I \in V_I,\ F_{SFT}(v_I) = Vrai \tag{2.6}$$

Cet ensemble n'est pas systématiquement unique. L'ensemble des sous-ensembles de transitions simultanément franchissables $SSFT$ est donc introduit comme-ci :

$$SSFT = \{SFT \subseteq T_{valid} \mid \exists v_I \in V_I,\ F_{SFT}(v_I) = Vrai\} \tag{2.7}$$

Soit le Grafcet présenté en Figure 1.10 et les étapes 2 et 3 actives. Les transitions $t2$ et $t3$ sont toutes deux validées. Dans cette situation, l'ensemble des sous-ensembles de transitions simultanément franchissables contient deux éléments $SSFT = \{SFT_1, SFT_2\}$, $SFT_1 = \{t_2, t_3\}$ dont la relation $F_{SFT_1} = a.b.a = a.b$ est vérifiée pour $v_I = (a = Vrai, b = Vrai)$ et $SFT_2 = \{t_3\}$ dont la relation $F_{SFT_2} = \overline{a.b}.a = a.\overline{b}$ est vérifiée pour $v_I = (a = Vrai, b = Faux)$.

A partir de la formalisation de ces règles, il est maintenant possible de définir précisément ce que sont les localités et les évolutions d'un AL et comment ces éléments sont construits.

2.3.3.2 Définition des localités et des évolutions de l'AL à partir du Grafcet

Les ensembles de variables d'entrée et de sortie d'un automate des localités sont identiques à ceux du Grafcet dont il est issu. Seuls les ensembles des localités et des évolutions doivent donc être redéfinis.

Une localité $l = (S_{Act}, O_{Em}, C(V_I))$ est caractérisée à la fois par une situation du Grafcet, l'ensemble des sorties émises pour cette situation du Grafcet ainsi que par la condition de stabilité de cette localité. L'ensemble des localités L est donc défini à partir de l'ensemble des situations S_{Act} en balayant toutes les situations possibles du modèle de spécification Grafcet. L'ensemble des sorties émises est alors déterminé à partir des actions continues associées aux étapes actives pour la situation donnée ainsi qu'à partir de l'historique des étapes activées pour les actions mémorisées. Enfin, la condition de stabilité $C(V_I)$ est obtenue à partir des conditions de franchissement des évolutions partant de la localité active, comme l'exprime la relation (2.2).

Une évolution $evol = (l_U, l_D, E(V_I))$ est caractérisée par une localité amont, une localité aval et une condition de franchissement. Chaque évolution se détermine donc, à partir d'une situation du Grafcet associée à une localité, par l'étude de l'ensemble $SSFT$ des ensembles de transitions simultanément franchissables, correspondant chacun à une évolution. Pour chacun de ces ensembles $SFT \in SSFT$, la condition de franchissement de l'évolution associée vaudra :

$$\begin{aligned} E(V_I) &= F_{SFT}(V_I) \\ &= \prod_{\substack{t \in T_{valid} \\ t \in SFT}} F_t(V_I) \wedge \prod_{\substack{t \in T_{valid} \\ t \notin SFT}} \overline{F_t(V_I)} \end{aligned} \tag{2.8}$$

Cela signifie que pour l'automate des localités associé au Grafcet présenté en Figure 1.10, deux évolutions partent de la localité caractérisée par les étapes 2 et 3 du Grafcet actives et aucune sortie émise. En effet, une évolution est créée par SFT et deux SFT sont répertoriés pour cette situation : $SSFT = \{SFT_1, SFT_2\} = \{\{t_2, t_3\}, \{t_3\}\}$. Depuis la localité associée à la situation $S_{Act} = \{2, 3\}$ et $O_{em} = \{\varnothing\}$, il existe donc une évolution allant vers la localité associée à la situation $S_{Act} = \{4, 5\}$ et $O_{em} = \{O_1\}$ et étiquetée par l'expression $E = F_{SFT_1} = a.b$ et une évolution allant vers la localité associée à la situation $S_{Act} = \{2, 5\}$ et $O_{em} = \{O_2\}$ et étiquetée par l'expression $E = F_{SFT_2} = a.\overline{b}$. Selon la relation donnée en (2.2), la condition de stabilité de la localité associée à la

situation $S_{Act} = \{2, 3\}$ et $O_{em} = \{\varnothing\}$ vaut $C = \overline{a.b + a.\overline{b}} = \overline{a}$.

2.3.3.3 Algorithme de construction de l'AL

La construction de l'automate des localités issu d'un Grafcet G est décrite par l'algorithme 3. Cet algorithme définit les ensembles d'entrées et de sorties de l'AL, sa localité initiale, les sorties émises pour cette localité initiale et appelle une fonction $LocationCondition(l)$ détaillée dans l'algorithme 4.

L'objectif de la fonction $LocationCondition(l)$ est de calculer la condition de stabilité $C(V_I)$ pour une localité l. Cette condition est obtenue en effectuant une succession de tâches. Tout d'abord, il est nécessaire de déterminer l'ensemble des $SFTs$ pour la situation courante du Grafcet, réalisé par la fonction $CreateSFT(S_G)$. Puis, pour chacun des $SFTs$, l'évolution correspondante est créée de la manière suivante :

- Mettre à jour les valeurs des variables d'activité d'étape après franchissement simultané des transitions du SFT, réalisé par la fonction $StepUpdate(S_G, SFT)$.
- Mettre à jour les valeurs des variables de sortie dans cette nouvelle situation, réalisé par la fonction $EmitOut(S_G)$.
- Créer, si elle n'existe pas déjà, la localité aval de l'évolution définie par la nouvelle situation du Grafcet et par les valeurs des sorties, réalisé par la fonction $CreateLocation(S_G, V_O)$. Ajouter cette localité nouvellement créée à la liste des localités dont la condition de stabilité doit être calculée.
- Créer l'évolution définie par la localité étudiée, la localité aval définie au pas précédent et la condition de franchissement $E(V_I)$, opération réalisée par la fonction $CreateEvol(L, L, E(V_I))$.

Ensuite, il est nécessaire de déterminer l'ensemble des évolutions correspondant à un changement de valeur des variables de sortie sans franchissement de transition. Cela est possible dans le cas d'actions conditionnelles associées aux étapes actives. Le calcul de ces évolutions est proche de celui des évolutions associées à des franchissement de transitions. Tout d'abord, l'ensemble des couples $(E(V_I), V_O)$ composés d'une condition sur les variables d'entrée et des sorties émises sont calculés via la fonction $CreateEvolOut(S_G, V_O)$ à partir de la connaissance de la situation du Grafcet et des sorties émises dans la configuration considérée. Puis, pour chaque possibilité d'évolution, la localité aval est si

nécessaire créée et ajoutée à la liste des localités dont la condition de stabilité doit être calculée ainsi que l'évolution dont la condition de franchissement $E(V_I)$ est la condition calculée précédemment.

Une fois toutes les évolutions définies pour la localité, il est possible de déterminer l'expression de la condition de stabilité de cette localité en prenant le complément de la conjonction de toutes les conditions de franchissement des évolutions partant de cette localité comme décrit dans la relation (2.2). Dans le cas où la localité initiale est traitée, une nouvelle localité, copie de la localité initiale, est créée. L'évolution entre la localité initiale et la localité copie correspond à l'absence de changement de situation du Grafcet ; la localité initiale devient donc purement instable.

Cette opération était nécessaire afin d'être capable d'atteindre une situation stable dès la première application des entrées dans le cas d'une exécution avec recherche de stabilité. Elle est conservée pour la construction de l'AL afin d'harmoniser les deux structures d'automates avec ou sans recherche de stabilité.

Algorithme 3 Construction de l'automate des localités

function BUILDLA(Grafcet)
    $I_{AL} = V_I$;
    $O_{AL} = V_O$;
    $SetL = \varnothing$
    $Oem_{Init} = EmitOut(S_{init})$;
    $L_{init} = CreateLocation(S_{Init}, Oem_{Init})$;
    $L_{init}.C = LocationCondition(L_{init})$; ▷ Crée un ensemble de localités dont la condition de stabilité doit être définie
    while $SetL \neq \varnothing$ **do**
        $L = SetL.pop$; ▷ Extrait une des localités de l'ensemble
        $L.C = LocationCondition(L)$;
    end while
    retourne : Automate des localités
end function

2.3.3.4 Exemple

L'exécution de l'algorithme de construction de l'AL est illustrée à partir du modèle de spécification Grafcet donné en Figure 1.10. Pour la situation initiale, l'étape 1 est active et les sorties O_1 et O_2 sont toutes deux à $Vrai$.

La première étape de l'algorithme est donc de créer la localité initiale 0 à partir de l'information sur les étapes actives ainsi que sur les sorties émises. A partir de cette

Algorithme 4 Fonction déterminant l'expression de la condition de stabilité pour une localité

1: **function** LocationCondition(l) ▷ Après avoir défini toutes les évolutions possibles depuis une localité, détermine la condition de stabilité de cette localité
2: $ActSteps = l \,.\, S_{Act}$; ▷ Etapes actives du Grafcet associées à la localité considérée
3: $ActOem = l \,.\, O_{Em}$; ▷ Sorties émises associées à la localité considérée
4: $NotC = Faux$ ▷ Complément de $C(I_g)$
5: **for** $t \in T_G$ **do**
6: Calcul de la réceptivité associée à chaque transition validée
7: **end for**
8: $SSFT = CreateSFT((ActSteps))$
9: **for** $SFT \in SSFT$ **do** ▷ Determination des nouvelles localités atteintes pour chaque évolution
10: $NewS = StepUpdate(ActSteps, SFT)$;
11: $E = F_{SFT}$;
12: $NewO = EmitOut(NewS)$;
13: **if** La localité aval existe déjà **then**
14: $CreateEvol(l, l_{new}, E)$; ▷ avec l_{new} la localité atteinte
15: **else** ▷ Crée la nouvelle localité et l'ajoute à l'ensemble des localité à traiter
16: $NewL = CreateLocation(NewS, NewO)$;
17: $SetL.append(NewL)$;
18: $CreateEvol(L, NewL, E)$;
19: **end if**
20: $NotC = NotC + E$; ▷ Met à jour la condition de stabilité de la localité considérée
21: **end for**
22: $SetEvolOut = CreateEvolOut(ActSteps, ActOem)$ ▷ Calcule les couples représentant un changement de sortie sans franchissement de transition
23: **for** $(E, O) \in SetEvolOut$ **do**
24: **if** La localité aval existe déjà **then**
25: $CreateEvol(l, l_{new}, E)$; ▷ avec l_{new} la localité atteinte
26: **else** ▷ Crée la nouvelle localité et l'ajoute à l'ensemble des localité à traiter
27: $NewL = CreateLocation(ActSteps, O)$;
28: $SetL.append(NewL)$;
29: $CreateEvol(L, NewL, E)$;
30: **end if**
31: $NotC = NotC + E$; ▷ Met à jour la condition de stabilité de la localité considérée
32: **end for**
33: $C = \overline{NotC}$;
34: **if** $l = L_{Init}$ **then** ▷ Si la localité est la localité initiale, crée une copie de cette localité ainsi qu'une évolution étiquetée par la condition de stabilité
35: $NewL = CreateLocation(ActSteps, ActOem)$;
36: $SetL.append(NewL)$;
37: $CreateEvol(l, NewL, C)$;
38: **retourne :** $\varnothing$
39: **else** ▷ Sinon, enregistre la bonne valeur pour la condition de stabilité
40: **retourne :** C
41: **end if**
42: **end function**

première localité créée, il est maintenant possible d'explorer les évolutions possibles afin de déterminer les localités suivantes. de plus, une copie de cette localité est créée pour le cas où la première évolution n'entrainerait pas de changement de situation.

Depuis cette situation initiale, l'ensemble des transitions simultanément franchissables SFT n'est composé que de $t1$ et sa réceptivité vaut $F_{t1} = a.\overline{b} + \overline{a}.b$. Le franchissement de cette transition conduit à la situation $NewS = \{2, 3\}$ où aucune sortie n'est émise $NewO = \varnothing$. La nouvelle localité 2 ainsi que l'évolution y menant sont créées.

Comme, depuis la situation initiale, aucune autre évolution ne peut avoir lieu, la condition de stabilité de cette localité $C = a.b + \overline{a}.\overline{b}$ est associée à l'évolution vers la localité 1 correspondant à la même situation que la localité initiale.

La fonction *LocationCondition(l)* est ensuite relancée pour les localités 1 et 2 créées précédemment afin de déterminer les évolutions dont elles sont la localité amont ainsi que leur condition de stabilité.

La Figure 2.5 présente le résultat obtenu après une première exécution de la fonction *LocationCondition(l)* et la Figure 2.6 présente le résultat final de construction de l'AL à partir de la spécification Grafcet.

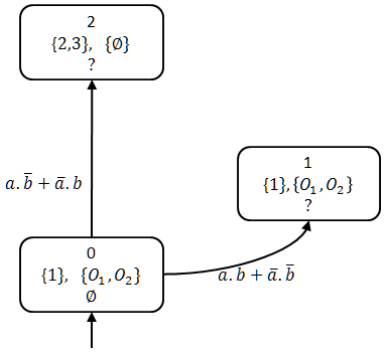


Figure 2.5 – Automate des localités partiel obtenu après une exécution de la fonction *LocationCondition(l)*

2.3.3.5 Discussion sur la complexité

Une borne supérieure du nombre de transitions ainsi que de situations du Grafcet, tout comme du nombre de localités et d'évolutions de l'AL peut être aisément calculée. Soit $s = |S_G|$ le nombre d'étapes du Grafcet et $n = |V_I|$ le nombre de variables d'entrée et $p = |V_O|$ le nombre de variables de sortie, les bornes sont définies par :

- $|T_G| = s * 2^n * 2^s$ transitions. Cette borne est atteinte si, depuis chaque étape du

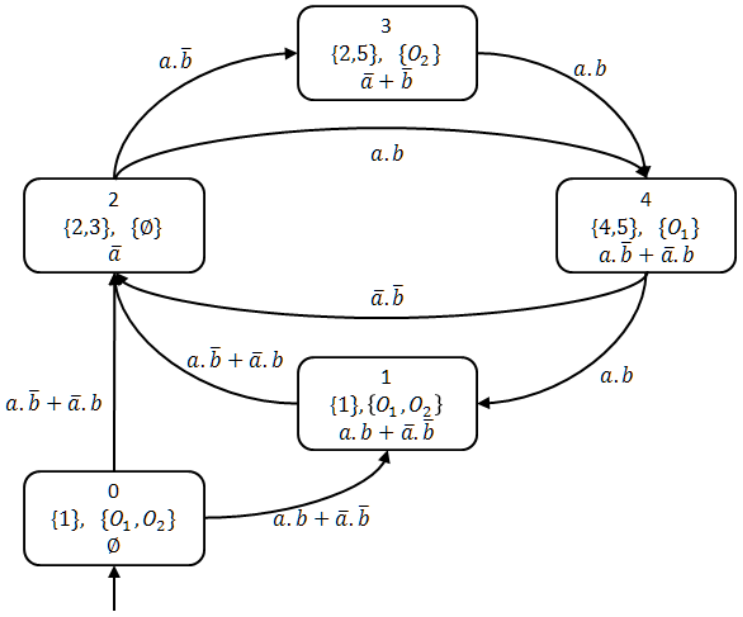


Figure 2.6 – AL déduit du Grafcet présenté Figure 1.10 avec une interprétation sans recherche de stabilité

Grafcet, il y a autant de transitions validées que de combinaisons possibles sur les valeurs des variables d'entrée et sur les valeurs des variables d'activité d'étape.

- $|L| = 2^s * 2^p$ localités. Cette borne est atteinte dans le cas où toutes les combinaisons d'étape actives sont possibles et que dans chaque étape, toutes les sorties peuvent être émises à travers une action conditionnelle.

- $|Evol| = 2^s * 2^p * 2^n$ évolutions. Cette borne est atteinte si, depuis chaque localité, il y a autant d'évolutions que de combinaisons possibles des variables d'entrée.

Néanmoins, il convient de remarquer que ces bornes sont extrêmement pessimistes. Les modèles de spécification Grafcet sont des représentations compactes de lois de commande qui ont été déterminées selon l'expertise d'ingénieurs automaticiens. Cela se traduit par exemple par le fait que les réceptivités associées aux transitions ne correspondent généralement pas à une unique mais à un ensemble de combinaisons de variables d'entrée.

2.4 Comparaison des modèles formels obtenus avec les deux interprétations

Cette section a pour but de mettre en évidence quelles sont les conséquences structurelles de l'hypothèse de l'algorithme d'exécution du Grafcet sur la construction du modèle du contrôleur. Celles-ci seront illustrées à travers deux exemples de spécification Grafcet.

2.4.1 Premier exemple

A partir du modèle de spécification Grafcet proposé en Figure 1.10, les deux automates considérant un algorithme sans et avec recherche de stabilité sont rappelés en Figure 2.7

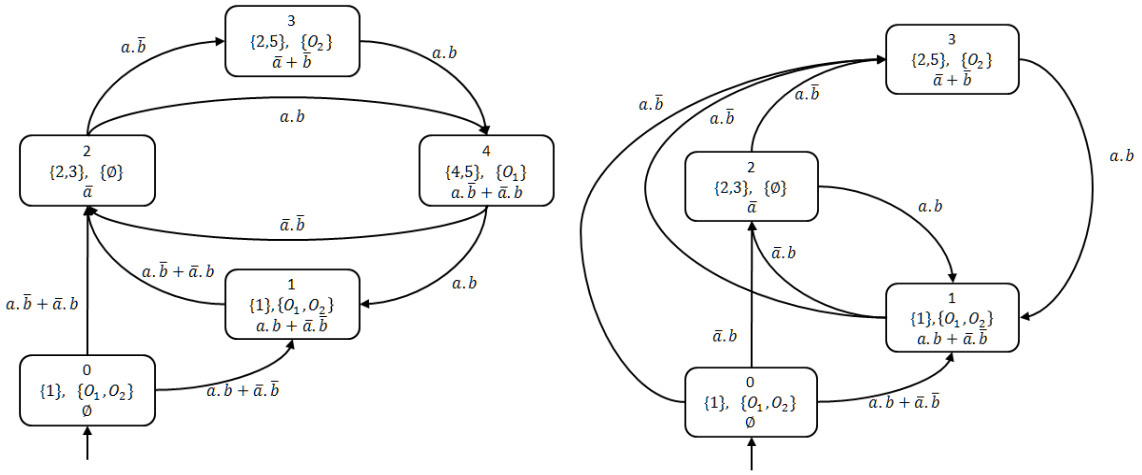


Figure 2.7 – AL présenté en Figure 2.6 et ALS présenté en Figure 2.3 construits à partir du Grafcet proposé en Figure 1.10

Ces deux figures mettent en évidence deux différences majeures entre les modèles :

- Le modèle représentant une interprétation sans recherche de stabilité contient une localité de plus que celui représentant une interprétation avec recherche de stabilité. Cette localité correspond à la situation $\{4, 5\}$ dans le modèle de spécification Grafcet. Cette situation est toujours instable à cause de la réceptivité de la transition t_5 qui vaut $FC = a.b$. En effet, la condition pour que les étapes 4 et 5 soient toutes deux actives est que les deux variables d'entrée a et b soient à $Vrai$; La condition de franchissement de la transition t_5 vaut alors toujours $F_{t_5} = Vrai$.

- Bien qu'il possède une localité de moins, et donc que les évolutions partant de cette

localité soient supprimées, le modèle de l'ALS contient autant d'évolutions que le modèle sans recherche de stabilité. Les évolutions ajoutées correspondent à des évolutions fugaces (séquences de franchissement de $SFTs$). Ce genre d'évolution existe uniquement dans l'interprétation avec recherche de stabilité.

La première remarque peut être généralisée à tout modèle de spécification. En effet, le nombre de localités de l'AL est toujours plus grand ou égal à celui des localités de l'ALS. Cela est dû au fait que seules les situations stables du Grafcet sont prises en compte dans le second cas. Par contre, aucune conclusion générale ne peut être donnée quant au nombre d'évolutions de l'AL par rapport à l'ALS. La présence d'évolutions fugaces crée des évolutions supplémentaires mais la suppression de localités purement instables supprime les évolutions partant de ces localités.

Ces deux automates apportent une bonne illustration de la remarque à propos de la complexité du modèle généré. En effet, pour le modèle de spécification Grafcet présenté Figure 1.10 (5 étapes et 2 variables d'entrée), les bornes supérieures sur le nombre de localités et d'évolutions sont respectivement $2^5 + 1 = 33$, correspondant à toutes les situations du Grafcet plus la localité initiale, et $33 * 2^2 = 132$; ces valeurs sont bien plus élevées que celles obtenues après construction de l'automate (5 et 8 pour l'AL et 4 et 8 pour l'ALS).

Cependant, ces modèles ne sont que des modèles intermédiaires utilisés pour obtenir une machine de Mealy. Comme présenté en sous-section 2.2.2, les deux machines équivalentes à l'AL et à l'ALS sont générées et présentées respectivement en Figure 2.8 et en Figure 2.9. Les grandeurs caractéristiques des différents modèles sont données dans la Table 2.1.

	Automate des localités		Machine de Mealy	
Recherche de stabilité	Localités	Evolutions	Etats	Transitions
Avec	4	8	3	12
Sans	5	8	4	16

Tableau 2.1 – Comparaison des modèles construit selon le type d'interprétation du Grafcet donné en Figure 1.10

Plusieurs informations peuvent être extraites de cette table :

- Le nombre d'états de la machine de Mealy est égal au nombre de localités de l'automate des localité moins un. Cette réduction du nombre d'états est due au

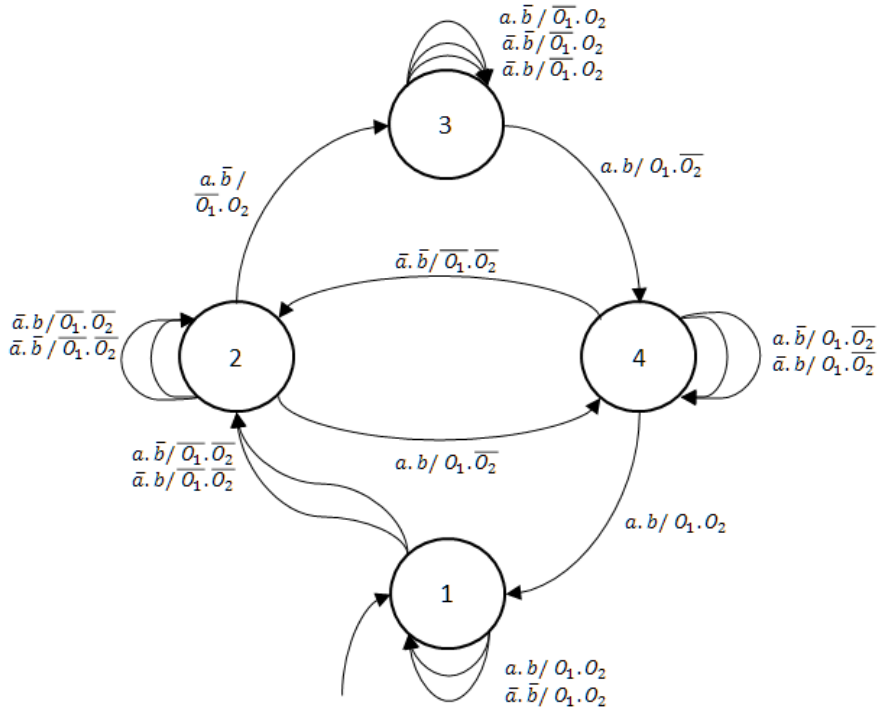


Figure 2.8 – Machine de Mealy correspondant à une interprétation sans recherche de stabilité issue de l'ALS en Figure 2.6

fait que l'état initial n'est pas associé à la localité initiale mais à la localité atteinte après évolution selon les valeurs initiales des variables d'entrée. La localité initiale, devenue non-représentative, peut ainsi être supprimée. Dans le cas de l'exemple présenté, avec pour combinaison de valeurs des variables d'entrée $a.b$, l'état initial correspond à la localité 1.

- Le nombre de transitions de la machine de Mealy est égal à $t = m * 2^n$ où m est le nombre d'états et n le nombre de variables d'entrée. Cela s'explique par le fait que depuis chaque état de la machine de Mealy il y a autant de transitions qu'il y a de combinaisons possibles sur les variables d'entrée.

2.4.2 Second exemple

Le modèle de spécification est cette fois-ci un modèle plus conséquent présenté Figure 2.10 et tiré de Provost et al. (2011b). Ce modèle contient 9 variables d'entrée $V_I = \{CS, on, mh, mb, z, a, b, dp, v\}$ et 10 variables de sortie $V_O = \{ILR, ILG, ILO, VA, VB, VC, MT, MR, MP+, MP-\}$.

A partir de cette spécification, les automates correspondant à une exécution avec ou

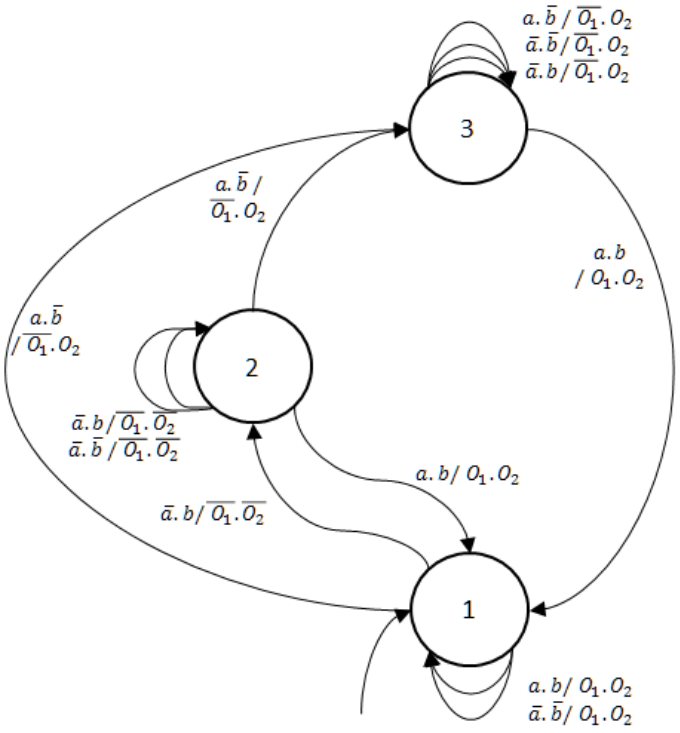


Figure 2.9 – Rappel de la Machine de Mealy correspondant à une interprétation avec recherche de stabilité présentée en Figure 2.4

sans recherche de stabilité sont construits. Les résultats sont affichés dans la Table 2.2.

	Automate des localités		Machine de Mealy	
Recherche de stabilité	Localités	Evolutions	Etats	Transitions
Avec	64	389	63	32256
Sans	67	370	66	33792

Tableau 2.2 – Comparaison des modèles construit selon le type d'interprétation du Grafcet donné en Figure 2.10

Il apparait que l'ALS contient 3 localités de moins que l'AL. Cela correspond à l'absence dans l'ALS des trois situations du Grafcet ($(F1, S40)$, $(A3, S40)$, $(A3, 1)$) qui sont purement instables. Il apparait aussi que l'ALS compte 19 transitions de plus que l'AL. Cela s'explique ainsi :

- L'absence des trois situations purement instables entraine la suppression de 7 évolutions pour l'ALS. En effet, depuis la situation instable $(F1, S40)$, sont supprimées l'évolution depuis $(F1, 42)$ et les évolutions vers $(F1, 1)$ et $(A3, 1)$; depuis la situation instable $(A3, S40)$, sont supprimées l'évolution depuis $(A3, 42)$ et l'évolution vers $(A3, 1)$; enfin, depuis la situation instable $(A3, 1)$, sont supprimées l'évolution depuis $(F1, 1)$ et l'évolution vers $(A1, 1)$.

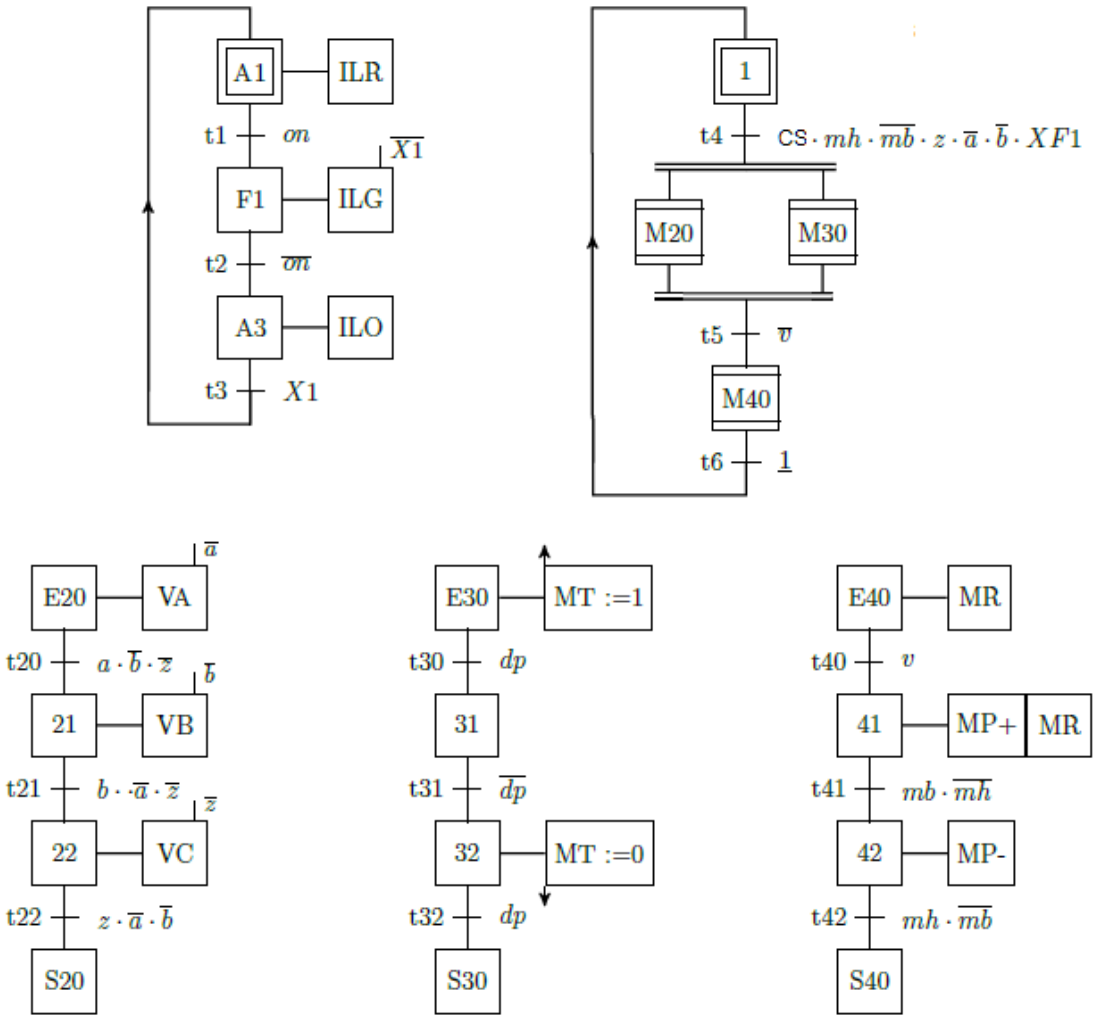


Figure 2.10 – Spécification Grafcet avec macro-étapes, actions conditionnelles et actions mémorisées

- L'existence de 26 évolutions fugaces possibles pour ce Grafcet entraine la création de 26 évolutions supplémentaires pour l'ALS par rapport à l'AL. On peut compter parmi elles l'évolution depuis $(F1, 42)$ vers $(F1, 1)$ représentant le franchissement fugace des transitions $t42$ et $t6$ lorsque $v_I = \{mh = Vrai, mb = Faux\}$, ou encore l'évolution depuis $(A1, 1)$ vers $(F1, E20, 31)$ représentant le franchissement fugace des transitions $t1$, $t4$ puis $t30$ lorsque $v_I = \{CS = Vrai,\ on = Vrai,\ mh = Vrai,\ mb = Faux,\ z = Vrai,\ a = Faux,\ b = Faux,\ dp = Vrai\}$.

2.5 Synthèse

Après un rappel des deux interprétations du Grafcet et de la méthode de construction du modèle de spécification pour une interprétation avec recherche de stabilité, ce chapitre

propose une extension de cette méthode pour le cas où le Grafcet est interprété sans recherche de stabilité. Le principe, les algorithmes de construction et un exemple y sont présentés. C'est sur ce modèle de spécification que pourront être comparées les observations effectuées sur le contrôleur logique exécutant le programme de commande.

La dernière partie du chapitre se focalise sur la mise en évidence des différences entre deux modèles formels de spécification issus du même Grafcet mais utilisant deux interprétations différentes. Hors, ces modèles sont les modèles formels représentant le comportement du programme de commande tel que décrit dans la spécification. Ils sont donc voués à être utilisés comme modèles de référence afin de juger de la validité de l'implantation de ce programme de commande dans un contrôleur logique. La présentation des différences entre les modèles formels en fonction de l'interprétation du Grafcet choisie met alors bien en évidence l'importance de considérer l'interprétation qui a effectivement été utilisée pour construire le programme de commande implanté dans l'API. En effet, si l'interprétation du modèle de référence diffère de celle utilisée pour le programme implanté, l'opération de validation peut conduire à des verdicts erronés ou biaisés qui sont à proscrire.

Chapitre 3

Contribution au test de conformité : définitions de relations de conformité

Sommaire

Introduction

Bien que Provost (2011) propose une méthode pour réaliser un test de conformité sur un contrôleur logique, aucune définition formelle d'une relation de conformité adapté à cet objectif n'est formellement définie. Ce mémoire présente donc un premier couple de relations de conformité adaptées aux machines de Mealy correspondant à des exécutions de Grafcet avec recherche de stabilité puis sans recherche de stabilité. En accord avec le choix des séquences SIC (Single-Input-Change) présenté dans Provost et al. (2010), cette relation fait l'hypothèse de l'absence de phénomène de lecture asynchrone d'évènements synchrones.

Cependant, le phénomène de lecture asynchrone d'évènements synchrones ne peut pas être totalement évité. Comme les relations de conformité utilisées actuellement ne sont pas capables de traiter ces cas et renvoient un verdict de non-conformité biaisé, une seconde relation de conformité pour chaque type d'exécution est définie. Cette seconde relation a pour but de détecter les cas de lecture asynchrone d'évènements synchrones. Comme cela correspond à une évolution des entrées/sorties non attendue, la transition pour laquelle ce cas a lieu ne peut pas être validée mais ne doit pas pour autant être déclarée non-conforme. Cette nouvelle relation de conformité autorisera donc à re-tester cette transition lorsque ce phénomène a lieu et est détecté. Bien que cette solution entraine des traitements supplémentaires sur la séquence de test, elle garantit l'absence de verdicts biaisés tout en ne violant pas l'objectif de test.

Le principal intérêt de ces relations de conformité est qu'elles sont non pas basées sur une comparaison unique des valeurs de sortie attendues et observées, que ce soit au cycle suivant l'application du pas de test ou après un nombre de cycles minimum, mais sur l'analyse d'une séquence d'observation des valeurs de sortie sur un nombre de cycle API déterminé, et ce, pour chaque pas de test. Cette approche est novatrice par rapport aux relations de conformité déjà existantes dans le sens où elle prend en compte les phénomènes ayant lieu lors de l'exécution de la séquence de test sur un contrôleur réel et non pas sur un modèle théorique du programme de commande qu'il exécute.

Ces travaux ont fait l'objet d'une communication en conférence internationale (Guignard and Faure (2014a)).

Le chapitre se décompose en quatre sections :

- La première section présente un rappel sur la génération et l'exécution des sé-

quences de test pour le test de conformité selon les travaux présentés dans Provost (2011). Elle met en évidence les avantages et inconvénients des différents types de séquences de test présentés.

- La seconde section présente la première contribution au test de conformité sous la forme de deux relations de conformité adaptées aux séquences de test SIC pour des interprétations du Grafcet avec ou sans recherche de stabilité. Ce couple de relations de conformité est défini car il représente une introduction simple aux relations de conformité adaptées aux séquences de test MIC.

- La troisième section présente la contribution majeure de ce chapitre qui consiste en la définition de relations de conformité adaptées aux séquences de test MIC pour des interprétations du Grafcet avec ou sans recherche de stabilité. Ce jeu de relation de conformité associée à une séquence de test MIC est une solution permettant d'obtenir une technique de test garantissant à la fois l'objectif de test et un verdict non biaisé.

- La quatrième section propose une synthèse des résultats obtenus pour le test de conformité.

3.1 Rappels sur la génération et l'exécution de séquences de test

3.1.1 Séquences de test SIC et MIC

La séquence de test est supposée générée selon la méthode du tour de transition Naito and Tsunoyama (1981), méthode de construction hors ligne d'un chemin résolvant le problème du postier chinois Kwan (1962). Les deux critères de construction sont :

- L'objectif de test : franchir au moins une fois toutes les transitions de la machine de Mealy.

- Le critère d'optimisation : Avoir la séquence la plus courte vérifiant l'objectif de test.

Selon Provost (2011), la séquence de test, notée TS, se compose d'une séquence de pas de test élémentaires et qui sont définis comme suit :

$$et = (s_u, i_M, s_d, o_M) \text{ où } \begin{cases} s_u \in S_M \text{ est l'état actif} \\ i_M \in I_M \text{ est la valeur des entrées appliquée} \\ s_d = \delta(s_u, i_M) \text{ est l'état aval de la transition} \\ o_M = \lambda(s_u, i_M) \text{ est la valeur des sorties attendue} \end{cases} \tag{3.1}$$

Cette séquence de test construite à partir de l'objectif de test de franchir au moins une fois toutes les transitions est souvent de taille très importante. En effet, si p est le nombre d'états de la machine de Mealy et $n = |V_I|$ le nombre de variables d'entrée, cette séquence sera de longueur au moins $p * 2^n$. Sachant que cette borne inférieure est optimiste car il est souvent nécessaire de franchir plusieurs fois certaines transitions du modèle formel de spécification pour atteindre des transitions non-encore franchies. Le problème de la taille de la séquence de test, même avec un critère d'optimisation visant à la minimiser, ne peut donc pas être négligé.

Une solution à ce problème consiste à utiliser les connaissances du fonctionnement du modèle Grafcet dont est issue le modèle formel de spécification afin de réduire la taille de la séquence de test. En effet, pour rappel, les franchissements de transition dans un Grafcet pouvaient se faire selon des évolutions fugaces (plusieurs franchissements successifs de transitions sans changement de valeur des variables d'entrée) ou de manière non-fugace (franchissement unique d'un ensemble de transitions menant à un état stable). Dans tous les cas, l'absence de cycle d'instabilité signifie qu'une évolution fugace ou non conduit toujours le Grafcet dans une situation telle qu'aucun nouveau franchissement de transition ou changement de variable de sortie n'aura lieu sans changement de valeur des variables d'entrée. Cela se traduit dans l'automate des localités (stable ou non) par l'existence de la condition de stabilité pour chaque localité qui se traduit elle-même dans la machine de Mealy par l'existence de self-loop sur les états (voir section 2.2). La Figure 3.1 illustre ce raisonnement.

Ainsi, pour toute transition de la machine de Mealy menant à un état stable, c'est-à-dire qui est associée à une situation stable pour le Grafcet, il existe une self-loop sur cet état portant la même étiquette. Il est donc possible, si le pas de test visant à tester cette transition est maintenu pendant deux cycles API, de tester à la fois la transition menant à cet état et la self-loop sur cet état portant la même étiquette que la transition.

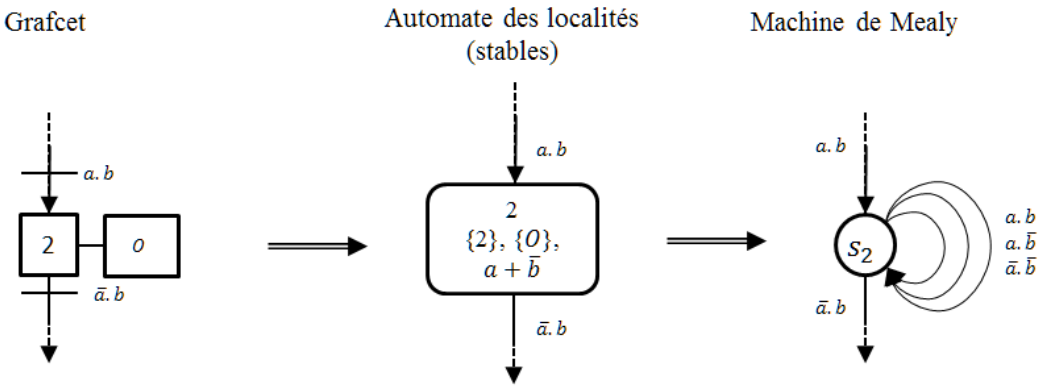


Figure 3.1 – D'une situation stable du Grafcet aux self-loop de la machine de Mealy

Cette stratégie est largement utilisée pour réduire le nombre de pas de la séquence de test. A partir de l'illustration donnée en Figure 3.1, il apparait que la transition menant à l'état s_2 et une des self-loop portent bien la même étiquette comme mis en évidence sur la Figure 3.2. et la stratégie proposée peut y être appliquée.

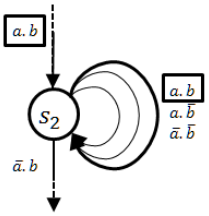


Figure 3.2 – Transition menant à un état stable et self-loop associée

La séquence de test de longueur minimale construite selon la méthode donnée par Naito and Tsunoyama (1981), vérifiant l'objectif de test et le critère d'optimisation pour l'exemple présenté en Figure 3.3, qui ne contient que des états stables, est :

$$\begin{aligned} TS = &((s_1, \overline{a}.\overline{b}, s_1, \overline{o}), (s_1, a.b, s_2, o), (s_2, \overline{a}.\overline{b}, s_2, o), \\ &(s_2, a.\overline{b}, s_1, \overline{o}), (s_1, a.b, s_2, o), (s_2, \overline{a}.b, s_1, \overline{o})) \\ &= (et_1, ..., et_6) \end{aligned} \tag{3.2}$$

Cette séquence de test est complète bien qu'elle ne contienne que six pas de test. En effet, même si la même transition est testée pour les pas de test et_2 et et_5, la stratégie définie précédemment est appliquée trois fois. Ainsi, pour les pas de test et_2, et_4 et et_6, deux transitions (la transition explicitée dans le pas de test puis la self-loop portant la même étiquette) sont testées pour chacun des pas de test.

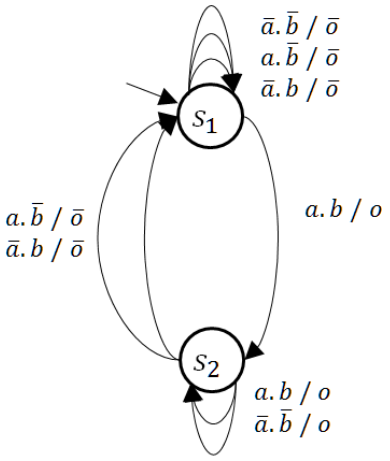


Figure 3.3 – Rappel de la machine de Mealy présentée en Figure 1.11

Cependant, les pas de test et_2 et et_3 voient plusieurs valeurs de variables d'entrée changer simultanément. En effet, pour et_2, les deux variables d'entrée a et b passent simultanément de $Faux$ à $Vrai$ et pour et_3 elles passent toutes deux simultanément de $Vrai$ à $Faux$. Ces pas de test sont dits MIC (Multiple-Input-Changes) et les séquences de test contenant des pas de test MIC sont elles aussi appelées séquence MIC. La Figure 3.4 présente le chronogramme représentant les changements de valeurs de 2 variables d'entrée a et b pour une succession de pas de test MIC. Or, comme il a été vu dans la section 1.1.3, ces changements simultanés de valeur de variables d'entrées, qui sont des évènements synchrones, peuvent être perçus de manière asynchrone par l'API et conduire à des verdicts biaisés lors de l'exécution de la séquence de test comme mis en évidence dans Provost (2011).

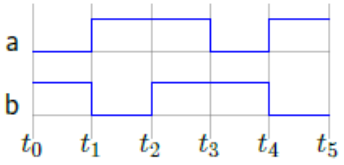


Figure 3.4 – Exemple de séquences MIC, issu de Provost (2011)

Une alternative proposée dans Provost et al. (2014) consiste à éviter l'occurrence de changements simultanés de valeur de variables d'entrée. Pour cela, le critère d'optimisation est changé. Plutôt que de rechercher la séquence de test qui soit de longueur minimale, un nouveau critère d'optimisation est défini telle que la séquence de test doive être la plus courte possible qui n'intègre que des pas de test ne nécessitant que le chan-

gement de valeur d'une seule variable d'entrée. En opposition avec les pas de test MIC, ces pas de test sont dits SIC. Une séquence de test ne contenant que des pas de test SIC est aussi dite séquence SIC. La Figure 3.5 présente un chronogramme représentant une séquence SIC composée de 2 variables d'entrée a et b.

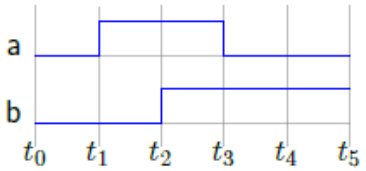


Figure 3.5 – Exemple de séquences SIC, issu de Provost (2011)

La séquence de test construite à partir de ce nouveau critère d'optimisation est alors :

$$\begin{aligned} TS_{SIC} =&((s_1, \overline{a}.\overline{b}, s_1, \overline{o}), (s_1, a.\overline{b}, s_1, \overline{o}), (s_1, a.b, s_2, o), \\ &(s_2, a.\overline{b}, s_1, \overline{o}), (s_1, a.b, s_2, o), (s_2, \overline{a}.b, s_1, \overline{o})) \end{aligned} \tag{3.3}$$

On peut observer que la séquence de test fait la même longueur que celle donnée dans la relation (3.2). Cependant, celle-ci n'est pas complète. En effet, même en appliquant la stratégie visant à tester à la fois les transitions menant à un état et les self-loop sur cet état, seules 7 transitions sont testées sur la machine de Mealy. La self-loop sur l'état 2 et étiquetée par $\overline{a}.\overline{b}$ n'est pas testée car il n'est pas possible d'atteindre cette situation par des changements de valeur des variables d'entrée SIC. En effet, les seules combinaisons de variables d'entrée permettant un pas de test SIC sont $\overline{a}.b$ et $a.\overline{b}$ et, depuis l'état 2, ils correspondent tous deux à un franchissement de transition menant à l'état 1. L'objectif de test n'est donc pas respecté avec cette séquence de test.

Le tableau 3.1 présente de manière synthétique les avantages ainsi que les inconvénients des séquences MIC et SIC.

	Avantages	Inconvénients
Séquence MIC	Objectif de test garanti	Risque de lecture asynchrone d'évènements synchrones menant à un verdict biaisé
Séquence SIC	Absence de risque de lecture asynchrone d'évènements synchrones	Objectif de test non forcément garanti

Tableau 3.1 – Comparatif entre des séquences de test SIC et MIC

Cette section étant un rappel des résultats publiés dans Provost (2011), elle ne considère qu'une interprétation du Grafcet avec recherche de stabilité. Lorsque l'interprétation du Grafcet est sans recherche de stabilité, les évolutions fugaces du Grafcet se traduisent dans la machine de Mealy par l'absence de la self-loop mise en évidence en Figure 3.2. L'Annexe présente une discussion sur la construction de la séquence de test dans ce cas.

3.1.2 Exécution d'une séquence de test

La procédure choisie pour l'exécution de la séquence de test dans Provost (2011) se décompose en trois temps :

- Dans un premier temps, les valeurs des variables d'entrée sont changées pour correspondre à la valeur définie pour le pas de test considéré.
- Ensuite, le banc de test attend la stabilisation des valeurs des variables de sortie. Cette attente est déterminée par une durée arbitrairement choisie qui doit être suffisamment longue pour passer tous les changements transitoires de valeur de sortie. Dans Provost (2011), cette durée est de 10 cycles API.
- Une fois la durée de stabilisation passée, le banc de test enregistre la valeur des variables de sortie. Cet valeur enregistrée est alors comparée à celle attendue selon le modèle de spécification. Si les valeurs sont différentes, alors le contrôleur logique est déclaré non-conforme à la spécification.

Le verdict rendu sur la conformité d'un contrôleur logique repose donc sur deux éléments :

- La comparaison d'une unique valeur des variables de sortie avec celle attendue pour chaque pas de test.
- L'observation de la valeur des variables de sortie après un temps arbitrairement choisi pour garantir la stabilisation de ces variables de sortie.

Cependant, le Tableau 3.1 montre que les deux types de séquence de test envisagées présentent toutes deux des inconvénients majeurs. Afin de contribuer au test de conformité de contrôleurs logique, il convient de traiter l'un des verrous présentés pour obtenir

une technique de test capable de générer une séquence et de l'exécuter en garantissant l'objectif de test ainsi que le verdict rendu.

Le fait que l'objectif de test ne puisse être garanti pour les séquences de test SIC est rédhibitoire et ne peut pas être réglé sans introduire des pas de test MIC. Une solution est donc de développer une relation de conformité qui soit capable de traiter des séquences de test MIC sans rendre de verdict biaisés. C'est-à-dire qui soit capable de détecter des phénomènes de lecture asynchrone d'évènements synchrones et de ne plus en conclure la non-conformité du contrôleur logique.

Il apparait alors qu'une simple comparaison directe entre la sortie attendue et la sortie observée suite à l'application des entrées n'est plus suffisante. En effet, comme illustré sur la Figure 3.6, ce phénomène se déroule sur deux cycles API. Pour le détecter, il est donc nécessaire d'observer le comportement du contrôleur pendant au moins 2 cycles pour chaque pas de test. **La relation de conformité ne doit donc pas être basée sur une comparaison ponctuelle des valeurs de sortie observées et attendues, mais sur l'analyse d'une séquence de valeurs de sortie observées pour chaque pas de test.**

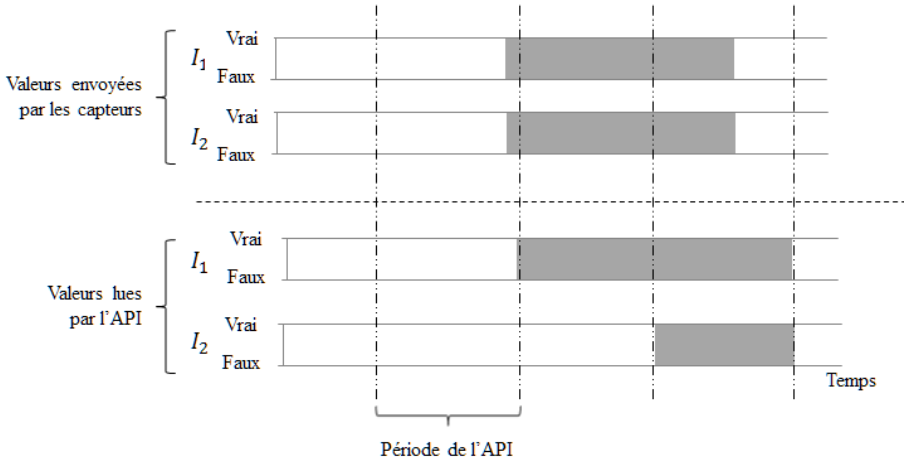


Figure 3.6 – Rappel de la Figure 1.6 présentant le phénomène de lecture asynchrone d'évènements synchrones

Cela signifie que la séquence de test telle qu'elle est définie actuellement n'a plus de sens. En effet, une séquence de test n'a pour but que de prévoir la succession de valeurs des variables d'entrée à appliquer aux bornes de l'API. Les franchissements de transitions de la machine de Mealy et le verdict en fonction des sorties observées ne concerne que la relation de conformité, et non pas la séquence de test.

Ainsi, pour la suite de ces travaux, la séquence de test est réduite à une seule séquence d'entrée :

$$\mathcal{TS} = (I_1, ..., I_n) \text{ avec } (I_1, ..., I_n) \in I_M^n \tag{3.4}$$

Dans un soucis de compréhension, il est choisi dans la suite de ce chapitre de traiter de la définition de relations de conformité selon un ordre croissant de complexité, depuis l'analyse de séquences de test SIC avec interprétation avec recherche de stabilité jusqu'à des séquences de type MIC avec interprétation sans recherche de stabilité. En accord avec les pratiques industrielles et les considérations présentées précédemment, ce dernier cas est celui préconisé pour la réalisation d'un test de conformité.

3.2 Relation de conformité adaptée aux séquences SIC

Cette partie propose une première définition de relations de conformité adaptée aux séquences de test SIC et capable de traiter des spécification Grafcet interprétées avec ou sans recherche de stabilité. Contrairement aux relations définies jusqu'à maintenant, celle-ci se base sur une séquence d'observation des variables de sortie pour chaque pas de test et non pas sur une unique observation, effectué juste après l'application des variables d'entrée ou après un temps déterminé.

Il est pour l'instant supposé que la séquence de test est de type SIC, c'est-à-dire qu'il n'y a pas de phénomène possible de détection asynchrone d'évènements synchrones. Cela permet donc de traiter un premier cas simplifié avant de traiter le cas des séquences de type MIC.

3.2.1 Interprétation avec recherche de stabilité

La relation de conformité analyse une séquence d'observation, et non pas la première (Tretmans (1996)) ou la dernière (Provost et al. (2011a)) observation de cette séquence. Cela signifie que pour chaque pas de test, le banc de test va enregistrer une séquence composée de valeurs de variables de sorties observées. Afin de pouvoir reproduire les franchissements de transitions sur le modèle formel de spécification, il est nécessaire d'effectuer un relevé des valeurs des variables de sortie pour chaque cycle API. Le nombre d'éléments dans une séquence d'observation est le nombre de pas d'observation, c'est-à-dire le nombre de relevé des valeurs des variables de sortie.

Définir une telle relation de conformité nécessite dans un premier temps de bien comprendre comment est réalisée l'exécution d'une séquence de test. Celle-ci se décompose en trois phases pour chaque pas de test :

- Le banc de test envoie la combinaison de valeurs d'entrée prévue pour le pas de test considéré aux bornes de l'API. Cette combinaison est maintenue tout le long du pas de test.
- L'API initialise un nouveau cycle et lit les valeurs des variables d'entrée à ses bornes. Il calcule et met à jour ses variables de sortie en conséquence pour le cycle courant ainsi que pour les suivants.
- Le banc de test observe tous les changements des valeurs de sortie émises par l'API.

Pour cela, et afin de traiter un cas générique, le cas d'une exécution d'un pas de test dans un contrôleur logique dont le modèle de spécification partiel est la machine de Mealy présentée en Figure 3.7 est étudiée.

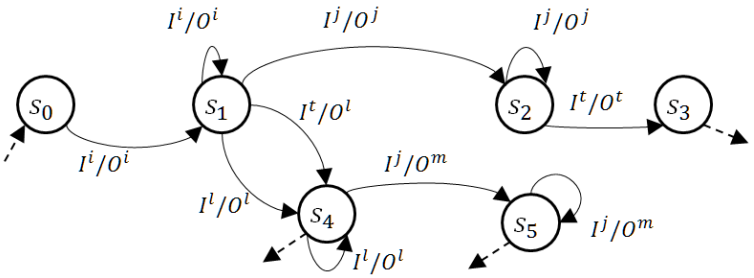


Figure 3.7 – Machine de Mealy générique partielle

Il est supposé que l'état s_1 est actif et que la combinaison de valeurs des variables d'entrée est I^i. Le banc de test applique ensuite la combinaison des variables d'entrée I^j afin de tester la transition allant de s_1 vers s_2. La Figure 3.8 illustre les évolutions des différentes variables pendant l'exécution de ce pas de test. Cette figure se décompose comme suit :

- Le chronogramme 3.8a représente l'évolution de la valeur des variables d'entrée aux bornes de l'API.
- Le chronogramme 3.8b représente la valeur des variables d'entrée lue par l'API

- Le chronogramme 3.8c représente l'évolution de la valeur des variables de sortie aux bornes de l'API.

On remarque l'apparition d'un décalage possible entre l'application des nouvelles valeurs de variable d'entrée et l'observation des nouvelles valeurs de variable de sortie. En effet, il existe un délai entre l'application des nouvelles valeurs des variables d'entrée et le moment où elles sont lues. De même, il existe un délai, dû au temps de traitement par l'API entre le moment où les variables d'entrée sont lues et le moment où les sorties sont mises à jour. Chacun de ces délais est inférieur ou égal à un temps de cycle API, mais les deux cumulés peuvent induire un décalage d'entre un et deux cycles API entre l'envoi des valeurs des variables d'entrée et la mise à jour des variables de sortie (voir sections 1.1.3.1 et 1.1.3.4), comme illustré sur la Figure 3.8. Comme la phase d'observation des valeurs des variables de sortie commence au cycle suivant l'envoi des nouvelles valeurs des variables d'entrée, il est nécessaire d'une part d'enregistrer au moins deux pas d'observation pour garantir que l'évolution attendue a bien eu lieu et d'autre part de tolérer l'absence d'évolution sur le premier pas d'observation.

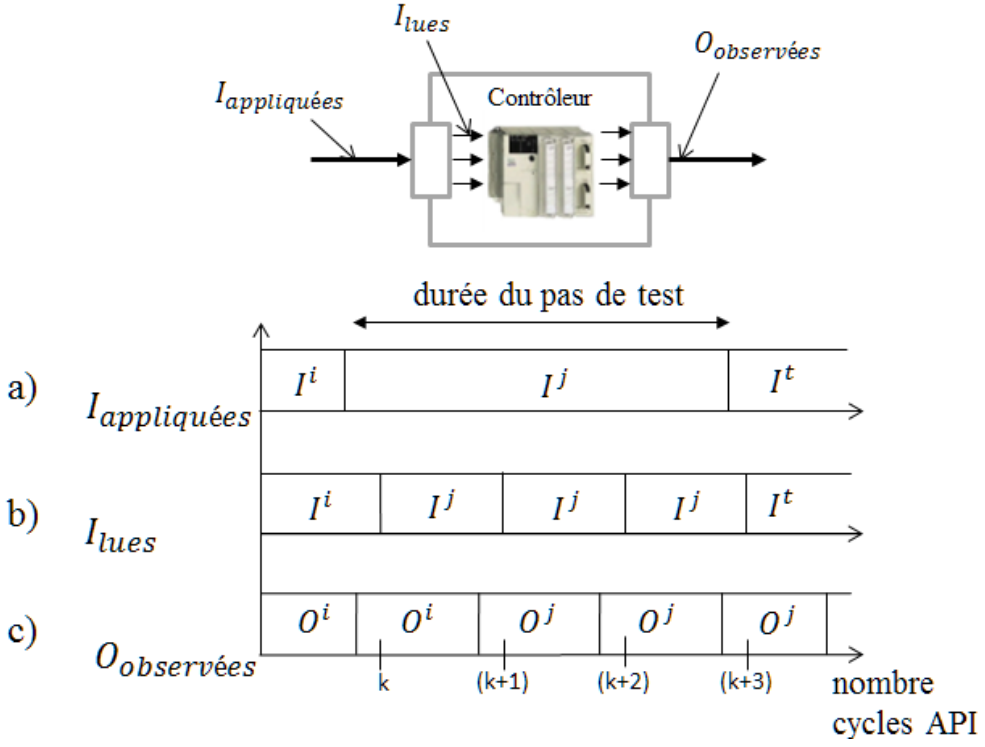


Figure 3.8 – Valeur des variables appliquées, lues et observées durant un pas de test

De plus, la stratégie permettant de tester deux transitions par pas de test, la transition à tester ainsi que la self-loop de l'état aval à cette transition portant la même

condition sur les variables d'entrée, implique qu'à partir du moment où la transition est effectivement franchie, on enregistre au moins une observation supplémentaire afin de tester également la self-loop traduisant la stabilité.

Cela signifie donc que pour chaque pas de test, le banc de test doit enregistrer au moins trois pas d'observation afin de rendre un verdict complet sur ce pas de test. Afin de garantir une durée minimale de la campagne de test, la longueur de la séquence d'observation est choisie égale à 3 pour chaque pas de test.

3.2.1.1 Définition

Une séquence d'observation conforme peut être de deux formes selon le cas de figure :

- Soit le phénomène de retard, tel que défini dans la section 1.1.3.4, a lieu, alors le premier pas d'observation sera la valeur des variables de sorties pour le pas de test précédent, suivi de deux occurrences de la valeur des variables de sortie attendue. La séquence d'observation est alors $O_{obs} = (O^i, O^j, O^j)$ dans le cas de l'illustration.

- Soit le phénomène de retard n'a pas lieu, et dans ce cas les trois pas d'observation doivent avoir la valeur des variables de sortie attendue, ce qui correspond au franchissement de la transition puis à deux franchissements successifs de la même self-loop traduisant la stabilité de l'état. La séquence d'observation est alors $O_{obs} = (O^j, O^j, O^j)$ dans le cas de l'illustration, qui est le cas présenté dans la Figure 3.8.

Formellement, l'étude de la séquence d'observation pour chaque pas de test est décrite dans la Définition 3.1.

Definition 3.1. *Relation de conformité pour une interprétation avec recherche de stabilité*

Soit s_c l'état courant de la machine de Mealy, soit $I^i \in I_M$ et $O^i \in O_M$ les valeurs courantes des variables d'entrée et de sortie,

Soit l'évènement d'entrée I^j,

Le contrôleur logique est dit conforme à la spécification pour ce pas de test si l'une de

ces relations est vérifiée :

$$\begin{cases} O_{Obs} = (O^i, O^j, O^j) & \\ ou & \text{avec } O^j \text{ tel que } \lambda(s_c, I^j) = O^j \\ O_{Obs} = (O^j, O^j, O^j) & \end{cases}$$

Le nouvel état courant de la machine de Mealy est alors défini par $s_d = \delta(s_c, I^j)$.

Autrement dit, cette relation de conformité signifie qu'un programme de commande implanté dans un API est déclaré conforme à la spécification si la séquence de sortie observée pour chaque pas de test expérimental est composée d'une séquence d'observation de combinaisons de variables de sortie identique correspondant à la sortie attendue pour le pas de test courant, qui peut être précédée par une observation de la combinaison de variables de sortie correspondant à la situation stable avant application de ce pas de test.

3.2.1.2 Exemple d'exécution d'une séquence de test pour un contrôleur conforme

Cette relation est illustrée à travers l'exemple de modèle de spécification présenté Figure 3.9 en appliquant la séquence de test :

$$\mathcal{TS}_{SIC} = (a.b, a.\overline{b}, \overline{a}.\overline{b}, \overline{a}.b, a.b, \overline{a}.b, \overline{a}.\overline{b}, a.\overline{b}, a.b, \overline{a}.b, a.b) \tag{3.5}$$

Dans cet exemple, sous l'hypothèse de n'avoir qu'une observation par cycle API et suite à la discussion précédente, la longueur de la séquence d'observation est choisie à $n = 3$ pour chaque pas de test. La Table 3.2 présente les résultats de l'exécution de la séquence de test

Il apparait ici que les 11 pas de test sont déclarés conformes avec un phénomène de retard à la lecture des entrées présent pour les pas 5,6 et 8.

3.2.1.3 Exemple d'exécution d'une séquence de test pour un contrôleur non conforme

La détection de contrôleur logique non conforme est aussi illustrée à travers l'exemple de modèle de spécification présenté Figure 3.9 en appliquant la séquence de test donnée en relation (3.5).

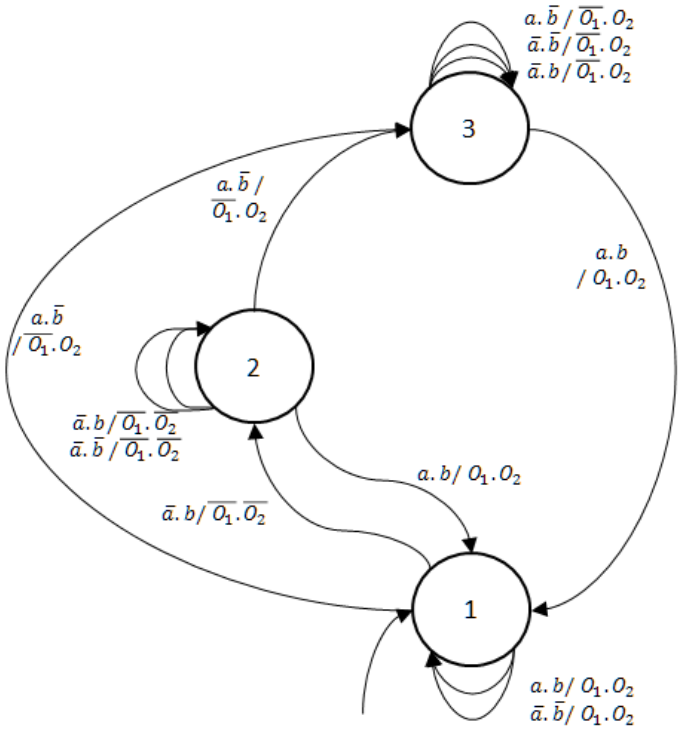


Figure 3.9 – Rappel de la Machine de Mealy correspondant à une interprétation avec recherche de stabilité présentée en Figure 2.4

La séquence observée est alors représentée dans la Table 3.3

Il apparait qu'au cinquième pas de test, la séquence observée $O_{obs} = (\overline{O_1}.O_2, O_1.\overline{O_2}, O_1.O_2)$ ne correspond à aucune des séquences attendues qui sont $(\overline{O_1}.O_2, O_1.O_2, O_1.O_2)$ et $(O_1.O_2, O_1.O_2, O_1.O_2)$. Le contrôleur logique testé est donc déclaré non conforme.

3.2.2 Interprétation sans recherche de stabilité

3.2.2.1 Définition

Comme défini en Annexe, lorsque le Grafcet est interprété sans recherche de stabilité, la séquence de test est construite à partir de l'ALS. Dans ce cas, lors du test d'une évolution fugace, un pas de test correspondant à une unique transition dans l'ALS se traduit par une succession de transitions, et donc d'émissions de sorties, dans le modèle du programme de commande. Par exemple, dans l'ALS présenté en Figure 3.9, la transition allant de l'état 1 à l'état 3 et étiquetée par $a.\bar{b}$ correspond au franchissement de deux transitions (de 1 vers 2 puis de 2 vers 3) sur le modèle de l'AL présenté en Figure 3.10.

Cela signifie que, à l'application de la valeur des variables d'entrée $a.\bar{b}$ sur l'API, celui-ci va émettre comme valeur de variable de sortie $\overline{O_1}.\overline{O_2}$ puis $\overline{O_1}.O_2$. La relation de

Pas de test		transition testée		Séquence d'observation	verdict
Numéro	Entrées	Etat amont	Etat aval		
1	$a.b$	1	1	$(O_1.O_2, O_1.O_2, O_1.O_2)$	OK
2	$a.\overline{b}$	1	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
3	$\overline{a}.\overline{b}$	3	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
4	$\overline{a}.b$	3	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
5	$a.b$	3	1	$(\overline{O_1}.O_2, O_1.O_2, O_1.O_2)$	OK
6	$\overline{a}.b$	1	2	$(O_1.O_2, \overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2})$	OK
7	$\overline{a}.\overline{b}$	2	2	$(\overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2})$	OK
8	$a.\overline{b}$	2	3	$(\overline{O_1}.\overline{O_2}, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
9	$a.b$	3	1	$(\overline{O_1}.O_2, O_1.O_2, O_1.O_2)$	OK
10	$\overline{a}.b$	1	2	$(\overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2})$	OK
11	$a.b$	2	1	$(\overline{O_1}.O_2, O_1.O_2, O_1.O_2)$	OK

Tableau 3.2 – Exécution de la séquence de test (3.5) par un contrôleur conforme

conformité doit donc s'adapter à ce cas de figure.

Elle se découpe ainsi en deux parties distinctes :

- Tout d'abord, une partie qui est similaire à la Définition 3.1 et qui concerne les pas de test associées à des évolutions non fugaces.

- Une seconde partie qui parcourt le modèle formel décrivant une interprétation sans recherche de stabilité lorsque l'évolution est fugace. Pour cela, chaque pas de la séquence d'observation (correspondant à un cycle API) est analysé afin de s'assurer qu'il correspond bien à la bonne sortie émise pour la combinaison de valeurs des variables d'entrée considérée lors du franchissement successif des transitions. Cela signifie que la séquence de test doit être suffisamment longue pour observer tous les franchissement successifs des transitions de la machine de Mealy composant l'évolution fugace analysée.

Concernant la longueur de la séquence d'observation, il a été vu à travers le point précédent que la longueur de la séquence d'observation doit au moins être égale au nombre

Pas de test		transition testée		Séquence d'observation	verdict
Numéro	Entrées	Etat amont	Etat aval		
1	$a.b$	1	1	$(O_1.O_2, O_1.O_2, O_1.O_2)$	OK
2	$a.\overline{b}$	1	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
3	$\overline{a}.\overline{b}$	3	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
4	$\overline{a}.b$	3	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
5	$a.b$	3	1	$(\overline{O_1}.O_2, O_1.\overline{O_2}, O_1.O_2)$	KO
$\varnothing$					

Tableau 3.3 – Exécution de la séquence de test (3.5) par un contrôleur non conforme

maximal, noté m, de franchissements successifs de transitions lors du test d'évolutions fugaces. De plus, il faut prévoir un pas d'observation supplémentaire pour le risque de décalage d'un cycle à l'émission des sorties ainsi que pour tester la stabilité de l'état final. Pour chaque pas de test, il faut donc observer au moins $m + 2$ cycles API.

Une définition formelle de cette relation de conformité est :

Definition 3.2. *Relation de conformité pour une interprétation sans recherche de stabilité*

Soit s_c l'état courant de la machine de Mealy, soit $I^i \in I_M$ et $O^i \in O_M$ les valeurs courantes des variables d'entrée et de sortie

Soit m le nombre maximum de franchissements successifs de transitions lors d'une évolution fugace,

Soit l'évènement d'entrée I^j

L'implantation du programme de commande est dite conforme à la spécification pour ce pas de test si l'une de ces relations est vérifiée :

$$\begin{cases} O_{Obs} = (O^i, O^j, ..., O^j) & \\ ou & \text{avec } O^j \text{ tel que } \lambda(s_c, I^j) = O^j \\ O_{Obs} = (O^j, O^j, ..., O^j) & \end{cases}$$

Le nouvel état courant de la machine de Mealy est alors défini par $s_d = \delta(s_c, I^j)$.

ou

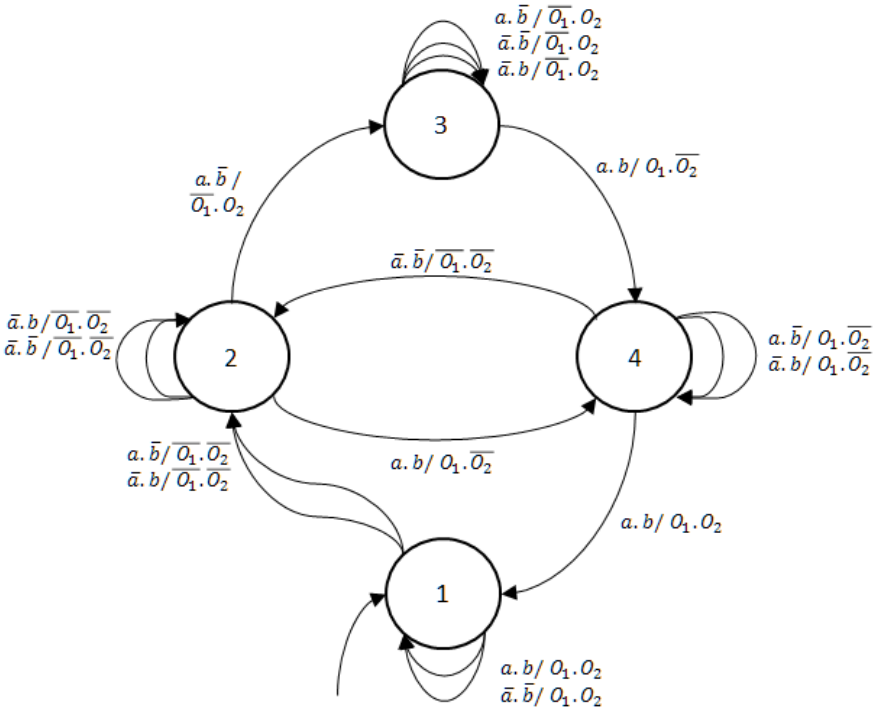


Figure 3.10 – Rappel de la machine de Mealy correspondant à une interprétation sans recherche de stabilité présentée en Figure 2.8

$$\begin{cases} O_{Obs} = (O^i, O_1, ..., O_{m+1}) \\ ou \\ O_{Obs} = (O_1, ..., O_{m+2}) \end{cases} \quad tel\ que \quad \begin{cases} \forall O_l, l = 1..m+1,\ on\ a : \\ \lambda(s_l, I^j) = O_l\ avec \\ s_0 = s_c\ et\ s_l = \delta(s_{l-1}, I^j) \\ et\ \lambda(s_m, I^j) = O_m = O_{m+1} \end{cases}$$

Le nouvel état courant de la machine de Mealy est alors défini par $s_d = s_m$.

3.2.2.2 Exemple d'exécution d'une séquence de test pour un contrôleur conforme

Cette partie présente une illustration de l'exécution d'une séquence de test sur un code implanté dans un API sans recherche de stabilité. Pour cela, le modèle de comportement du programme de commande considéré est celui représenté par la Figure 3.10 et la séquence de test à exécuter est donnée par la relation (3.5) :

$$\mathcal{TS}_{SIC} = (a.b, a.\overline{b}, \overline{a}.\overline{b}, \overline{a}.b, a.b, \overline{a}.b, \overline{a}.\overline{b}, a.\overline{b}, a.b, \overline{a}.b, a.b)$$

L'évolution fugace la plus longue impliquant le franchissement successif de 2 transitions, $2 + 2 = 4$ pas d'observation sont nécessaire pour chaque pas de test afin de

pouvoir rendre un verdict sur la conformité du contrôleur logique testé. Le résultat de l'exécution de la séquence de test est présenté dans la Table 3.4.

Le verdict de conformité est rendu pour tous les pas de test de cette exécution. On peut cependant noter que lors pas 6 et 8 un retard a été observé sur des évolutions non-fugaces, les pas 2 et 11 correspondent à des évolutions fugaces sur lesquelles aucun retard n'a eu lieu et les pas 5 et 9 correspondent à des évolutions fugaces pour lesquelles un retard a été observé.

3.3 Relation de conformité adaptée aux séquences MIC

La section 3.1 a mis en évidence les lacunes de la technique de génération et d'exécution de la séquence de test actuelle. Cette section ambitionne donc, pour les Grafcet interprétés avec ou sans recherche de stabilité, de proposer une relation de conformité qui permette l'exécution de séquences de test MIC définies formellement et assurant l'absence de verdicts biaisés.

3.3.1 Conséquences sur l'observation du phénomène de désynchronisation

De même que pour la définition des relations de conformité précédentes, il est nécessaire de comprendre comment, lors de l'exécution d'une séquence de test, le phénomène de lecture asynchrone d'évènements synchrones va impacter les informations traitées par le banc de test. Cela est illustré à travers l'analyse d'un pas de test appliqué à un contrôleur donc le modèle de spécification partiel est donné par la Figure 3.7.

Il est à nouveau supposé que l'état s_1 est actif et que la combinaison de valeurs des variables d'entrée est I^i. Le banc de test applique ensuite la combinaison des variables d'entrée I^j afin de tester la transition allant de s_1 vers s_2. On suppose qu'à l'application de cette combinaison, l'API ne perçoit qu'une partie des mises à jour des variables d'entrée et lit la combinaison I^l au cycle k. Au cycle suivant, il perçoit bien l'ensemble des variables d'entrée telles qu'envoyées par le banc de test. Les évolutions des variables envoyées, lues et émises sont représentées dans la Figure 3.11.

En considérant la séquence d'entrée lue par l'API, une implantation correcte émettra la séquence de sortie (O^i, O^l, O^m). Cette séquence correspond au franchissement successif

Pas de test		transition testée			Séquence d'observation	verdict
Numéro	Entrées	Etat amont	Etat interm.	Etat aval		
1	$a.b$	1	$\varnothing$	1	$(O_1.O_2, O_1.O_2,$ $O_1.O_2, O_1.O_2)$	OK
2	$a.\overline{b}$	1	2	3	$(\overline{O_1}.\overline{O_2}, \overline{O_1}.O_2,$ $\overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
3	$\overline{a}.\overline{b}$	3	$\varnothing$	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2,$ $\overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
4	$\overline{a}.b$	3	$\varnothing$	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2,$ $\overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
5	$a.b$	3	4	1	$(\overline{O_1}.O_2, O_1.\overline{O_2},$ $O_1.O_2, O_1.O_2)$	OK
6	$\overline{a}.b$	1	$\varnothing$	2	$(O_1.O_2, \overline{O_1}.\overline{O_2},$ $\overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2})$	OK
7	$\overline{a}.\overline{b}$	2	$\varnothing$	2	$(\overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2},$ $\overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2})$	OK
8	$a.\overline{b}$	2	$\varnothing$	3	$(\overline{O_1}.\overline{O_2}, \overline{O_1}.O_2,$ $\overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
9	$a.b$	3	4	1	$(\overline{O_1}.O_2, O_1.\overline{O_2},$ $O_1.O_2, O_1.O_2)$	OK
10	$\overline{a}.b$	1	$\varnothing$	2	$(\overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2},$ $\overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2})$	OK
11	$a.b$	2	4	1	$(O_1.\overline{O_2}, O_1.O_2,$ $O_1.O_2, O_1.O_2)$	OK

Tableau 3.4 – Exécution de la séquence de test (3.5) par un contrôleur conforme

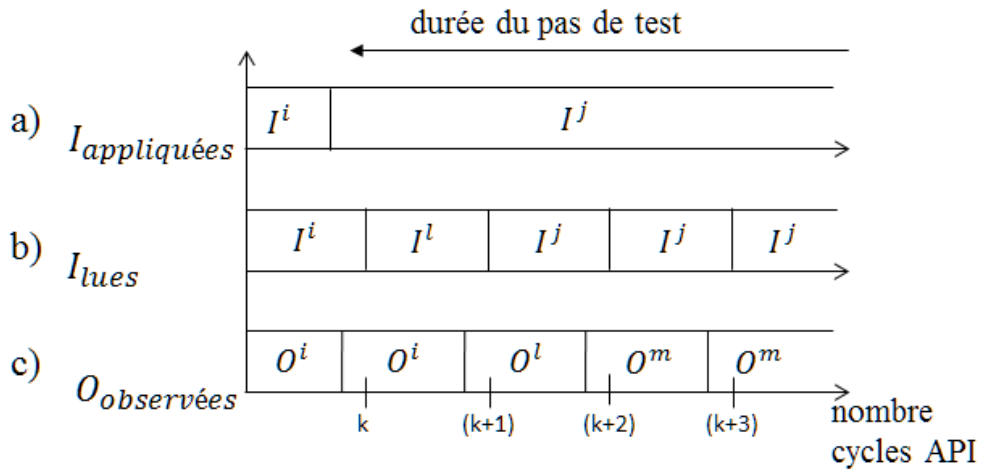


Figure 3.11 – Valeur des variables appliquées, lues et observées durant un pas de test où des évènements synchrones sont perçus de manière asynchrone

des transitions depuis l'état s_1 vers s_4 puis de l'état s_4 vers s_5. Comme la séquence de sortie initialement attendue est (O^i, O^j, O^j), la relation définie dans la section précédente rendra un verdict de non-conformité de cette implantation, verdict biaisé. Cette section propose une relation de conformité visant à empêcher l'émission de verdicts biaisés.

3.3.2 Interprétation avec recherche de stabilité

3.3.2.1 Définition

Le principe de cette nouvelle relation de conformité est non pas de se limiter à la comparaison entre la séquence d'observation avec une séquence attendue, comme défini dans la Définition 3.1, mais de prendre aussi en compte les cas où les variables d'entrée ne sont pas mises à jour de manière synchrone. Cette nouvelle relation est donc une extension de la première en ajoutant une phase de comparaison supplémentaire. Le but de cette phase est de vérifier si la séquence d'observation peut correspondre à un franchissement successif de transitions à partir de l'état courant du modèle de spécification tel que :

- Un éventuel premier pas d'observation traduisant le décalage d'un cycle API de la réponse à la sollicitation du banc de test.

- La première transition est étiquetée par une combinaison de variables d'entrée correspondant à seulement une partie des évènements sur les variables d'entrée menant de la valeur au pas de test précédent (I^i dans la section précédente) à la

valeur du nouveau pas de test (I^j dans la section précédente) ;

- La seconde transition est étiquetée par la combinaison de valeurs des variables d'entrée correspondant à celle appliquée par le banc de test (I^j dans la section précédente).

Une remarque doit cependant être formulée concernant la longueur de la séquence. Comme illustré dans l'exemple présenté, un phénomène de désynchronisation a des conséquences se mesurant en deux temps sur le modèle de comportement du programme de commande. Tout d'abord, la lecture partielle des entrées mises à jour entraine le franchissement d'une première transition puis, au pas de cycle suivant, la lecture des variables d'entrée est complète et cela entraine le franchissement d'une autre transition. Ajouté à cela les phénomènes de retard à l'observation des nouvelles sorties ainsi que le test sur la stabilité des états, le nombre de cycle API qui doivent être observés pour garantir l'acquisition de la totalité des évolutions passe de 3 à 4 cycles. Cela signifie que pour chaque pas de test, la séquence d'observation doit compter au moins 4 éléments.

La définition formelle qui traduit cette tolérance et qui respecte l'hypothèse d'une unique observation par cycle API est :

Definition 3.3. *Relation de conformité pour une interprétation avec recherche de stabilité et phénomène de désynchronisation toléré*
Soit s_c l'état courant de la machine de Mealy, soit $I^i \in I_M$ et $O^i \in O_M$ les valeurs courantes des variables d'entrée et de sortie,
Soit l'évènement d'entrée I^j,
L'implantation est dite conforme à la spécification pour ce pas de test si l'une de ces relations est vérifiée :

$$\begin{cases} O_{Obs} = (O^i, O^j, O^j) & \\ ou & \text{avec } O^j \text{ tel que } \lambda(s_c, I^j) = O^j \\ O_{Obs} = (O^j, O^j, O^j) & \end{cases}$$

Le nouvel état courant de la machine de Mealy est alors défini par $s_d = \delta s_c, I^j$.
Un phénomène de désynchronisation est détecté et la transition doit être re-testée si

l'une de ces relations est vérifiée :

$$\begin{cases} O_{Obs} = (O^i, O^k, O^{k+1}, O^{k+1}) \\ ou \\ O_{Obs} = (O^k, O^{k+1}, O^{k+1}, O^{k+1}) \end{cases} \quad avec \quad \begin{cases} O^k tel\ que\ \exists I^x \in I_M\ qui\ vérifie \\ (I^x \backslash I^i \cup I^i \backslash I^x) \subset (I^i \backslash I^j \cup I^j \backslash I^i) \\ \quad et\ \lambda(s_c, I^x) = O^k \\ et \\ O^{k+1} tel\ que\ \exists s \in S_M\ qui\ vérifie \\ \delta(s_c, I^x) = s\ et\ \lambda(s, I^j) = O^{k+1} \end{cases}$$

Cette relation de conformité se compose de deux parties distinctes. La première a la même structure que la relation donnée par la Définition 3.1 avec un pas d'observation supplémentaire. Si le verdict de cette première partie est négatif, une seconde analyse est réalisée. Les transitions partant de l'état amont (s_b) et qui sont étiquetées par des évènements d'entrée correspondant à une mise à jour partielle (I^x) entre la valeur du pas de test précédent (I^i) et celle du pas de test traité (I^j) sont répertoriées. La relation vérifie dans un premier temps si le franchissement de l'une de ces transitions entraine l'émission de la combinaison de valeurs des variables de sortie qui corresponde à la valeur au premier ou au second pas d'observation. Dans un second temps, et depuis l'état atteint sur franchissement de cette transition, la relation de conformité vérifie que la transition étiquetée par la valeur des entrées du pas de test (I^j) émet bien la valeur de sortie observée et que la stabilité est atteinte à la suite de ce nouveau franchissement de transition. dans ce cas, le verdict est que la transition doit à nouveau être testée.

En effet, si un phénomène de lecture asynchrone d'évènements synchrones a lieu, cela signifie que ce n'est pas la transition attendue, mais une autre transition du modèle de comportement du programme de commande qui a été franchie. Par conséquent, bien que l'implantation de ce programme de commande ne soit pas déclarée non conforme, la transition attendue n'a pas été franchie ni testée. Il est nécessaire de la re-tester afin de garantir l'objectif de test.

De manière formelle, les combinaisons de valeur de variables d'entrée correspondant à une mise à jour partielle des entrées se définit par la relation :

$$\left(I^x \backslash I^i \cup I^i \backslash I^x\right) \subset \left(I^i \backslash I^j \cup I^j \backslash I^i\right) \tag{3.6}$$

Où $I^1 \backslash I^2$ est l'ensemble des variables d'entrée de l'API qui ont pour valeur $Vrai$ dans I^1 et $Faux$ dans I^2. Ainsi, le terme $I^x \backslash I^i$ (respectivement $I^i \backslash I^x$) représente l'ensemble des variables mises à $Vrai$ entre I^i et I^x (resp. mises à $Faux$ entre I^i et I^x). L'union de ces deux ensembles représente l'ensemble des variables qui ont changé de valeur entre I^i et I^x. Comme seules les transitions correspondant à une lecture partielle des nouvelles valeurs des variables d'entrée sont associées à un phénomène de désynchronisation, la relation vérifie que cet ensemble de variables dont la valeur a changé est bien inclus dans l'ensemble de variables dont la valeur doit changer entre I^i et I^j, ce qui représente l'évolution qui aurait dû avoir lieu.

Ces notations peuvent être illustrées à travers l'exemple présenté en Figure 3.12. Si $I^i = a.b$ et $I^j = \overline{a}.\overline{b}$, alors les combinaisons de valeur de variables d'entrée qui correspondent à une lecture partielle des nouvelles valeurs sont $a.\overline{b}$ ainsi que $\overline{a}.b$. Plus généralement, si n variables d'entrée sont simultanément changées par le banc de test entre deux pas de test, alors $2^n - 2$ combinaisons correspondant à une lecture partielle des variables d'entrée peuvent être définies.

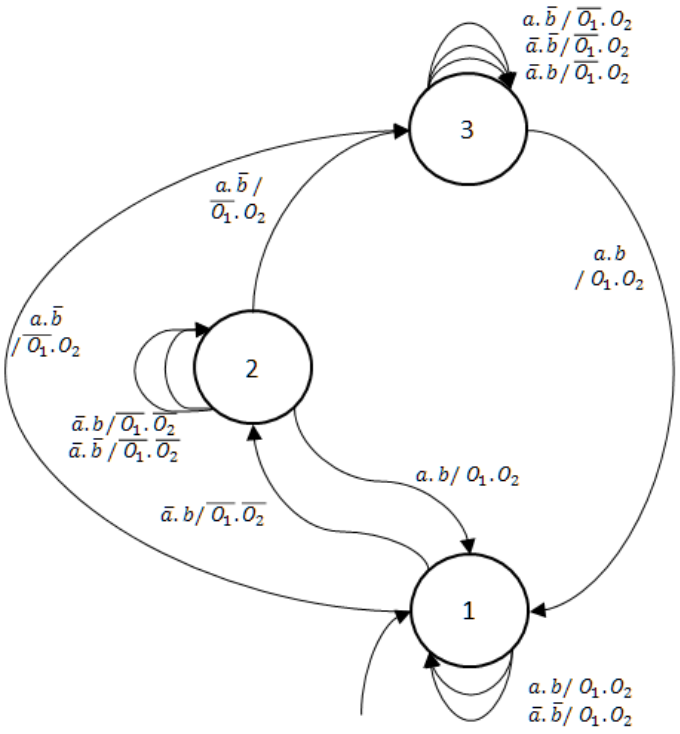


Figure 3.12 – Rappel de la Machine de Mealy correspondant à une interprétation avec recherche de stabilité présentée en Figure 2.4

3.3.2.2 Exemple d'exécution d'une séquence de test pour un contrôleur conforme

Cette relation est illustrée à travers l'exemple de modèle de spécification présenté Figure 3.9 en appliquant la séquence de test :

$$\mathcal{TS}_{MIC} = (a.b, a.\overline{b}, \overline{a}.\overline{b}, \overline{a}.b, a.b, \overline{a}.\overline{b}, \overline{a}.b, \overline{a}.\overline{b}, a.\overline{b}, a.b, \overline{a}.b, a.b) \tag{3.7}$$

On peut remarquer que cette séquence de test ajoute au sixième pas de test le test de la self-loop sur l'état 1 qui n'était pas testable via une séquence SIC. L'objectif de test est vérifié avec cette séquence.

Le résultat de l'exécution de la séquence de test est présenté dans la Table 3.5.

Pas de test		transition testée		Séquence d'observation	verdict
Numéro	Entrées	Etat amont	Etat aval		
1	$a.b$	1	1	$(O_1.O_2, O_1.O_2, O_1.O_2, O_1.O_2)$	OK
2	$a.\overline{b}$	1	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
3	$\overline{a}.\overline{b}$	3	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
4	$\overline{a}.b$	3	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
5	$a.b$	3	1	$(\overline{O_1}.O_2, O_1.O_2, O_1.O_2, O_1.O_2)$	OK
6	$\overline{a}.\overline{b}$	1	1	$(O_1.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	!
Désynchronisation détectée					

Tableau 3.5 – Exécution de la séquence de test (3.7) par un contrôleur conforme

Lors de l'exécution du sixième pas de test, il est attendu que le programme de commande continue d'émettre les valeurs de sortie $O_1.O_2$. En effet, ce pas de test vise à tester la self-loop sur l'état 1 étiquetée par la valeur de variables d'entrée $\overline{a}.\overline{b}$. Comme la séquence de sortie observée $O_{obs} = (O_1.O_2, O_1.\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$ ne correspond pas à une évolution à temps ou avec un retard d'un cycle API, cette séquence doit être

analysée afin de détecter une possible lecture asynchrone des évènements synchrones.

Dans le cas considéré :

- La première valeur de sortie observée $O_1.O_2$ peut être interprétée comme un retard d'un cycle.
- La seconde valeur de sortie observée est $\overline{O_1}.O_2$. Les deux transitions correspondant à une lecture partielle des évènements d'entrée $a.\overline{b}$ et $\overline{a}.b$ émettent respectivement comme valeur de variable de sortie $\overline{O_1}.O_2$ et $\overline{O_1}.\overline{O_2}$. La transition étiquetée par la combinaison de valeur de variable d'entrée $a.\overline{b}$ et dont l'état aval est l'état 3 émet la sortie observée. Cette transition est donc choisie et son état aval est considéré comme l'état actif pour l'évolution suivante.
- La troisième valeur de sortie observée est $\overline{O_1}.O_2$. La transition étiquetée par la valeur des variables d'entrée correspondant au pas de test $\overline{a}.\overline{b}$ est la self-loop sur l'état 3 et émet bien la sortie observée.
- Enfin, la dernière valeur observée $\overline{O_1}.O_2$ est compatible avec un maintien de la valeur des variables d'entrée et un nouveau franchissement de la self-loop sur l'état 3.

Ainsi, le phénomène de lecture asynchrone des évènements synchrones est détecté. Comme la transition à tester n'a à priori pas été franchie puisque les entrées n'ont pas été lues comme prévu, elle ne peut pas être validée ni déclarée invalide. Il est donc nécessaire de la tester à nouveau afin de garantir l'objectif de test.

3.3.2.3 Exemple d'exécution d'une séquence de test pour un contrôleur non conforme

La détection de contrôleur logique non conforme est aussi illustrée à travers l'exemple de modèle de spécification présenté Figure 3.12 en appliquant la séquence de test donnée en relation (3.7).

La séquence observée est alors représentée dans la Table 3.6

Dans ce cas, la séquence observée pour le sixième pas de test est analysée. Les deux dernières observations ne sont pas compatible avec une lecture correcte des évènements d'entrée. Cependant, la seconde observation de valeur des variables de sortie $O_1.O_2$ ne

Pas de test		transition testée		Séquence d'observation	verdict
Numéro	Entrées	Etat amont	Etat aval		
1	$a.b$	1	1	$(O_1.O_2, O_1.O_2, O_1.O_2, O_1.O_2)$	OK
2	$a.\overline{b}$	1	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
3	$\overline{a}.\overline{b}$	3	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
4	$\overline{a}.b$	3	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
5	$a.b$	3	1	$(\overline{O_1}.O_2, O_1.O_2, O_1.O_2, O_1.O_2)$	OK
6	$\overline{a}.\overline{b}$	1	1	$(O_1.O_2, O_1.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	KO
$\varnothing$					

Tableau 3.6 – Exécution de la séquence de test (3.7) par un contrôleur non conforme

correspond à aucune des sorties émises par les transitions correspondant à une lecture partielle des évènements d'entrée $a.\overline{b}$ et $\overline{a}.b$. La possibilité de lecture asynchrone d'évènements synchrone est donc écartée et le verdict de non-conformité est rendu.

3.3.3 Interprétation sans recherche de stabilité

3.3.3.1 Définition

Le principe de la relation de conformité prenant en compte les phénomènes de désynchronisation pour une interprétation sans recherche de stabilité est le même que dans le cas d'une interprétation avec recherche de stabilité. La seule différence tient dans le fait que cela peut entrainer des franchissements de transitions impliquées dans des évolutions fugaces.

Ainsi, la première partie de la relation de conformité est identique à celle donnée en Définition 3.2, validant l'implantation, suivie en cas d'échec de cette validation d'une partie visant à déterminer si un phénomène de lecture asynchrone d'évènements synchrones peut avoir eu lieu. Cette partie analyse la séquence d'observation comme suit :

- Un éventuel premier pas d'observation traduisant le décalage d'un cycle API de la réponse à la sollicitation du banc de test.

- La première transition est étiquetée par une combinaison de variable d'entrée qui correspond à une mise à jour partielle des variables d'entrée, comme il a été défini par la relation (3.6).

- La seconde transition est étiquetée par la combinaison des variables d'entrée correspondant à celle appliquée par le banc de test. Si cette transition fait partie d'une évolution fugace, alors le modèle de comportement du programme de commande évoluera jusqu'à atteindre la stabilité.

Il faut noter que même si la première transition, correspondant à une mise à jour partielle des entrées, fait partie d'une évolution fugace, seule cette transition sera franchie avant le changement de valeur des variables d'entrée. En effet, contrairement à ce qui a été évoqué en Annexe sur la génération d'une séquence de test lorsque l'interprétation est sans recherche de stabilité, ce n'est pas le banc de test qui impose le changement de valeur des variables d'entrée mais l'API lui-même par son cycle de fonctionnement.

La Définition 3.4 présente une définition formelle qui traduise cette description.

Definition 3.4. *Relation de conformité pour une interprétation sans recherche de stabilité et phénomène de désynchronisation toléré*
Soit s_c l'état courant de la machine de Mealy, soit $I^i \in I_M$ et $O^i \in O_M$ les valeurs courantes des variables d'entrée et de sortie,
Soit m le nombre maximum de franchissement successif de transitions lors d'une évolution fugace,
Soit l'évènement d'entrée I^j,
L'implantation est dite conforme à la spécification pour ce pas de test si l'une de ces relations est vérifiée :

$$\begin{cases} O_{Obs} = (O^i, O^j, ..., O^j) & \\ ou & \text{avec } O^j \text{ tel que } \lambda(s_c, I^j) = O^j \\ O_{Obs} = (O^j, O^j, ..., O^j) & \end{cases}$$

Le nouvel état courant de la machine de Mealy est alors défini par $s_d = \delta(s_c, I^j)$.
ou

$$\begin{cases} O_{Obs} = (O^i, O_1, ..., O_{m+1}) \\ ou \\ O_{Obs} = (O_1, ..., O_{m+2}) \end{cases} \quad tel\ que \begin{cases} \forall O_l, l = 1..m+1,\ on\ a : \\ \lambda(s_l, I^j) = O_k\ avec \\ s_0 = s_c\ et\ s_l = \delta(s_{l-1}, I^j) \\ et\ \lambda(s_m, I^j) = O_m = O_{m+1} \end{cases}$$

Le nouvel état courant de la machine de Mealy est alors défini par $s_d = s_m$.
Un phénomène de désynchronisation est détecté et la transition doit être retestée si l'une de ces relations est vérifiée :

$$\begin{cases} O_{Obs} = (O^i, O^k, O_1, ..., O_{m+1}) \\ ou \\ O_{Obs} = (O^k, O_1, ..., O_{m+2}) \end{cases} \quad avec \begin{cases} O^k\ tel\ que\ \exists I^x \in I_M\ qui\ vérifie \\ (I^x \backslash I^i \cup I^i \backslash I^x) \subset (I^i \backslash I^j \cup I^j \backslash I^i) \\ \ et\ \lambda(s_c, I^x) = O^k \\ et\ soit\ s_0 = \delta(s_c, I^x) \\ \forall O_l, l = 1..m+1,\ on\ a : \\ \lambda(s_l, I^j) = O_k\ avec\ s_l = \delta(s_{l-1}, I^j) \\ et\ \lambda(s_m, I^j) = O_m = O_{m+1} \end{cases}$$

De la même manière que pour le cas de l'exécution avec recherche de stabilité, le phénomène de désynchronisation va nécessiter un cycle API supplémentaire avant d'atteindre la stabilité du système. Il est donc désormais nécessaire d'observer $m+3$ cycles API pour chaque pas de test, m étant la longueur maximale d'évolution fugace en terme de franchissements successifs de transitions.

3.3.3.2 Exemple d'exécution d'une séquence de test pour un contrôleur conforme

Cette partie présente une illustration de l'exécution d'une séquence de test MIC sur un code implanté dans un API sans recherche de stabilité. Pour cela, le modèle de comportement du programme de commande considéré est celui représenté par la Figure 3.13 et la séquence de test à exécuter est donnée par la relation (3.7) :

$$\mathcal{TS}_{MIC} = (a.b, a.\overline{b}, \overline{a}.\overline{b}, \overline{a}.b, a.b, \overline{a}.\overline{b}, \overline{a}.b, \overline{a}.\overline{b}, a.\overline{b}, a.b, \overline{a}.b, a.b)$$

L'évolution fugace la plus longue impliquant le franchissement successif de 2 transitions, $2 + 3 = 5$ pas d'observation sont nécessaires pour chaque pas de test afin de

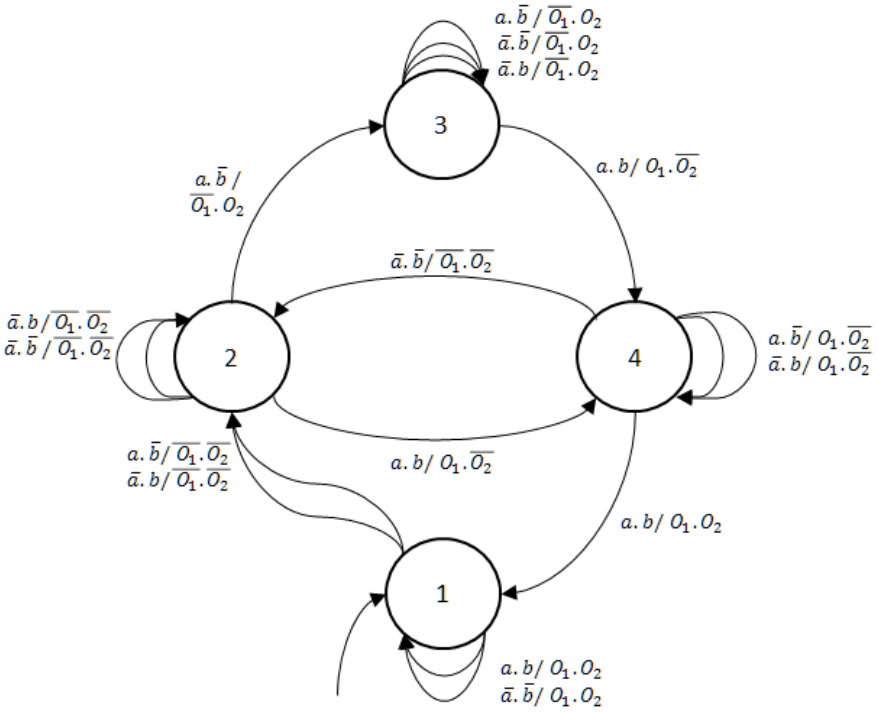


Figure 3.13 – Rappel de la machine de Mealy correspondant à une interprétation sans recherche de stabilité présentée en Figure 2.8

pouvoir rendre un verdict sur la conformité du contrôleur logique testé. Le résultat de l'exécution de la séquence de test est présenté dans la Table 3.7.

la séquence de sortie observée correspondant au sixième pas de test $O_{obs} = (O_1.O_2,$ $\overline{O_1}.\overline{O_2}$, $\overline{O_1}.\overline{O_2}$, $\overline{O_1}.\overline{O_2}$, $\overline{O_1}.\overline{O_2}$, $\overline{O_1}.\overline{O_2})$ ne correspond pas au franchissement attendu de la self-loop sur l'état 1 ; cette séquence doit être analysée afin de détecter une possible lecture asynchrone des évènements synchrones :

- La première valeur de sortie observée $O_1.O_2$ peut être interprétée comme un retard d'un cycle.
- La seconde valeur de sortie observée est $\overline{O_1}.\overline{O_2}$. Les deux transitions correspondant à une lecture partielle des évènements d'entrée $a.\bar{b}$ et $\bar{a}.b$ émettent toutes deux la valeur de sortie observée. Leur état aval étant l'état 2, cet état est considéré actif pour l'évolution suivante.
- La troisième valeur de sortie observée est $\overline{O_1}.\overline{O_2}$. La transition étiquetée par la valeur des variables d'entrée correspondant au pas de test $\bar{a}.\bar{b}$ est la self-loop sur l'état 2 et émet bien la sortie observée.
- Enfin, la dernière valeur observée $\overline{O_1}.O_2$ est compatible avec un maintien de la

Pas de test		transition testée			Séquence d'observation	verdict
Numéro	Entrées	Etat amont	Etat interm.	Etat aval		
1	$a.b$	1	$\varnothing$	1	$(O_1.O_2, O_1.O_2, O_1.O_2, O_1.O_2, O_1.O_2)$	OK
2	$a.\overline{b}$	1	2	3	$(\overline{O_1}.\overline{O_2}, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
3	$\overline{a}.\overline{b}$	3	$\varnothing$	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
4	$\overline{a}.b$	3	$\varnothing$	3	$(\overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2, \overline{O_1}.O_2)$	OK
5	$a.b$	3	4	1	$(\overline{O_1}.O_2, O_1.\overline{O_2}, O_1.\overline{O_2}, O_1.O_2, O_1.O_2)$	OK
6	$\overline{a}.\overline{b}$	1	$\varnothing$	1	$(O_1.O_2, \overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2}, \overline{O_1}.\overline{O_2})$	!
Désynchronisation détectée						

Tableau 3.7 – Exécution de la séquence de test (3.7) par un contrôleur conforme

valeur des variables d'entrée et un nouveau franchissement de la self-loop sur l'état 2.

Un phénomène de lecture asynchrone d'évènements synchrones est donc détecté et la transition est déclarée à re-tester.

3.3.4 Discussion sur la poursuite du test après détection de phénomène de détection asynchrone d'évènements synchrones

La Définition 3.3 ainsi que la Définition 3.4 permettent d'éviter de rendre des verdicts biaisés quelque soit la manière donc l'API perçoit les changements de valeur des variables d'entrée. Cependant, lorsqu'une séquence d'observation différente de celle attendue est acceptée par la relation de conformité, l'exécution de la séquence de test ne peut pas être poursuivie. En effet, cette séquence est construite hors-ligne avant son exécution. De plus, il a été vu dans les exemples traités qu'une lecture asynchrone d'évènements synchrones peut entrainer le modèle de comportement du programme de commande à atteindre un état qui n'est pas celui initialement prévu par la séquence de test. Poursuivre

son exécution ne permet donc plus de garantir l'objectif de test puisque la suite de la séquence est construire à partir d'un autre état que celui actif.

Trois solutions peuvent alors être considérées pour pallier à ce problème :

- Une séquence de test qui assure le franchissement de toutes les transitions qui n'ont pas encore été franchies (y compris celle où le phénomène de désynchronisation a eu lieu) est reconstruite à partir du nouvel état initial qui est l'état stable courant.
- La séquence de test initiale qui a été construite hors-ligne est conservée mais le prochain pas de test démarrera depuis le nouvel état actif
- Une séquence de test partielle est recalculée depuis le nouvel état actif afin de revenir à l'état d'avant l'exécution du pas de test qui a entrainé le phénomène de désynchronisation. La séquence de test initiale est ensuite reprise à partir du dernier pas de test exécuté.

3.3.4.1 Calcul d'une nouvelle séquence de test

Selon cette solution, une nouvelle séquence de test partant du dernier état atteint et qui assure le franchissement de toutes les transitions qui n'ont pas encore été franchies est construite. Cette solution est toujours possible et nécessite seulement de sauvegarder l'ensemble des transitions qui ont déjà été franchies.

Sur l'exemple de la séquence de test définie par la relation (3.7) basée sur le modèle de spécification présenté Figure 3.12, et en supposant l'exécution présentée en Table 3.5, les pas de test de et_1 à et_5 ont été correctement exécutés. Cependant, 6 transitions doivent encore être testées afin de vérifier l'objectif du test. La nouvelle séquence de test qui devra être exécutée par le banc de test part de l'état 3 avec pour valeur des variables d'entrée initiale $\overline{a}.\overline{b}$ s'écrit :

$$\mathcal{TS}_\alpha = (a.b, \overline{a}.\overline{b}, \overline{a}.b, \overline{a}.\overline{b}, a.\overline{b}, a.b, \overline{a}.b, a.b) \tag{3.8}$$

3.3.4.2 Repositionnement dans la séquence de test

Cette solution a l'avantage de ne nécessiter aucun calcul de nouvelle séquence de test pendant la phase d'exécution mais repose sur l'hypothèse qu'il est possible de trouver un pas de test déjà exécuté qui mène au même état actif.

Si cette hypothèse est vérifiée, la séquence de test est exécutée à nouveau à partir de ce pas de test (non inclus). Si une telle situation existe dans la séquence de test mais pour un pas de test qui n'a pas encore été exécuté, cette solution ne peut pas être retenue car certaines transitions peuvent ne pas être testées dans la suite de la séquence et l'objectif de test ne serait alors plus garanti.

Appliqué à l'exemple, l'hypothèse est vérifiée. En effet, l'état actif est 3, état atteint après exécution du troisième pas de test. La séquence de test est donc ré-exécutée à partir du quatrième pas de test inclus :

$$\mathcal{TS}_\beta = (a.b, \overline{a}.\overline{b}, \overline{a}.b, \overline{a}.\overline{b}, a.\overline{b}, a.b, \overline{a}.b, a.b) \tag{3.9}$$

on peut remarquer que les relations (3.8) et (3.9) sont identiques. Cependant, comme la seconde solution ne nécessite aucune opération de construction de séquence de test, elle est préférable à la première.

3.3.4.3 Retour au pas de test précédent

Cette solution peut être appliquée lorsque l'hypothèse définie dans la sous-section précédente n'est pas vérifiée. Elle consiste en la reconstruction d'une séquence de test SIC partielle qui mène de l'état actif vers l'état avant exécution du dernier pas de test. Ensuite, la séquence de test initiale pourra être reprise à partir du pas de test où a eu lieu le phénomène de désynchronisation. La condition de changement unique de variable d'entrée sur la séquence partielle est imposée afin d'éviter que, sur des pas de test MIC, un autre phénomène de lecture asynchrone d'évènements synchrones ait lieu et ne permette pas de rejoindre l'état avant exécution du pas de test. Cependant, il n'est pas toujours possible de construire une séquence SIC entre deux états d'une machine de Mealy ; cette solution n'est donc pas toujours possible.

Pour l'exemple considéré, la séquence SIC partielle peut être construite et la séquence de test complète à exécuter devient :

$$\mathcal{TS}_\delta = (a.\overline{b}, a.b, \overline{a}.\overline{b}, \overline{a}.b, \overline{a}.\overline{b}, a.\overline{b}, a.b, \overline{a}.b, a.b) \tag{3.10}$$

Il faut cependant noter que la séquence de test donnée par la relation (3.10) est plus longue que celles données par les relations (3.8) et (3.9) et nécessite le calcul de

nouveaux pas de test. Cette troisième solution n'est donc pas forcément la plus efficace.

3.4 Synthèse

Ce chapitre traite de la génération et de l'exécution de séquences de test pour le test de conformité de contrôleur logique à partir de modèle de spécification sous forme d'une machine de Mealy. Dans un premier temps, un rappel sur la technique et les limitations de la méthode développée dans Provost (2011) sont exposées. Les contributions de ce mémoire de thèse pour le test de conformité sont ensuite présentées sous la forme de deux ensembles de relations de conformité dont la caractéristique principale est d'être basées sur des séquences d'observation pour chaque pas de test plutôt qu'une observation unique des valeurs des variables de sortie.

La première est un jeu de relations de conformité adaptées aux séquences de test SIC pour des spécifications Grafcet interprétées avec ou sans recherche de stabilité. Bien qu'il ait été relevé que les séquences de test de type SIC peuvent ne pas être en mesure de garantir l'objectif de test, la définition formelle d'une relation de conformité adaptée à ce type de séquence permet à la fois de combler un manque dans les travaux précédents et de faire office d'introduction pour une relation de conformité plus complexe adaptée aux séquences de test de type MIC.

La seconde contribution est un jeu de relations de conformité adaptées aux séquences de test MIC pour des spécifications Grafcet interprétées avec ou sans recherche de stabilité. Cette relation de conformité est capable d'analyser et de détecter des phénomènes de lecture asynchrone d'évènements synchrones en plus des comportements conformes. Cela permet donc de garantir l'absence de verdicts biaisés dus à une lecture asynchrone d'évènements synchrones et complète la technique de test développée dans Provost (2011).

La Table 3.8 synthétise l'intérêt de ces nouvelles relations de conformité.

	Avantages	Inconvénients
Séquence SIC	Absence de risque de lecture asynchrone d'évènements synchrones	Objectif de test non forcément garanti
Séquence MIC sans relation	Objectif de test garanti	Risque de lecture asynchrone d'évènements synchrones menant à un verdict biaisé
Séquence MIC avec relation	Objectif de test garanti Absence de verdicts biaisés	OK

Tableau 3.8 – Synthèse

Chapitre 4
Contribution à la validation en boucle fermée

Sommaire

Introduction

Dans la problématique de validation de contrôleurs logiques, les travaux présentés dans ce mémoire visent à proposer des solutions qui permettent de valider le contrôleur lorsqu'il est couplé à une partie opérative, c'est-à-dire lorsqu'il se trouve dans sa configuration d'utilisation finale. C'est pourquoi ce chapitre propose de développer une nouvelle méthode de validation de contrôleur logique dont le modèle de spécification est sous la forme d'une machine de Mealy : la validation en boucle fermée. Comme cette approche est novatrice, les phases d'observation et d'analyse des séquences observées sont présentées dans ce chapitre. Une hypothèse sur l'observation des variables d'entrée et de sortie est tout d'abord choisie afin de simplifier les algorithmes d'analyse, puis cette hypothèse est levée afin de garantir la non-invasivité de la technique pour le contrôleur. Le développement de cette technique de validation a été fait suivant deux axes différents. Le premier, basé sur des techniques d'enforcement, dont les résultats ont fait l'objet d'une communication en conférence internationale Guignard and Faure (2013b) et le second, basé sur des techniques d'identification, dont les résultats ont aussi fait l'objet d'une communication en conférence internationale Guignard and Faure (2014b). Le second axe de recherche ayant donné des résultats plus probants pour la validation en boucle fermée, lui seul sera présenté dans ce mémoire.

Ce chapitre est organisé comme suit :

- La première section présente le principe de la validation en boucle fermée ainsi que son intérêt en comparaison avec le test de conformité puis décrit la phase d'observation des entrées et sorties du contrôleur relié à la partie opérative commandée. Une solution est notamment proposée à l'un des verrous majeurs de ce type de technique qui est le manque de connaissance sur la taille totale du modèle à analyser, et par conséquent la durée de cette phase d'observation nécessaire pour rendre un verdict sur la validité du contrôleur.

- La deuxième section s'intéresse à l'algorithme qui permet la validation d'un contrôleur logique relié à sa partie opérative lorsque les entrées et sorties sont observées à l'intérieur du contrôleur. Cette configuration est invasive pour le contrôleur car elle nécessite de modifier le programme de commande mais elle permet de mettre en place une méthode assez simple pour une première approche.

- La troisième section lève la restriction imposée dans la section précédente en proposant une technique de validation de contrôleur logique en boucle fermée lorsque l'observation des entrées-sorties est réalisée aux bornes du contrôleur. Ainsi, aucune modification du programme de commande ni des interfaces de communication du contrôleur n'est nécessaire et le verdict rendu est garanti pour le contrôleur logique qui sera utilisé dans le système de contrôle/commande.

- La quatrième section propose enfin une synthèse des travaux réalisés pour la validation en boucle fermée de contrôleur logique.

4.1 Validation en boucle fermée

4.1.1 Principe de la validation en boucle fermée

Contrairement au test de conformité, la validation en boucle fermée n'isole pas le contrôleur de la partie opérative mais observe le comportement de l'ensemble couplé. Ainsi, le système de contrôle/commande va fonctionner sans intervention extérieure et les valeurs des variables d'entrée et de sortie doivent être extraites du système en boucle fermée. A l'aide des séquences de valeurs de variables d'entrée et de sortie observées et en supposant la partie opérative exempte de fautes, il est alors possible de construire un modèle du système en boucle fermé. Ce modèle peut ensuite être analysé et comparé avec le modèle de la spécification afin de rendre un verdict sur la validité du contrôleur. La figure 4.1 illustre cette technique.

Cette méthode possède l'avantage d'offrir une validation du contrôleur logique dans une configuration d'utilisation réelle sans requérir de modèle de la partie opérative qui introduirait des imprécisions dans le comportement. Cependant, il peut parfois être dangereux, surtout pour des systèmes critiques, de coupler un contrôleur logique non encore validé à sa partie opérative. Une solution est donc de coupler le contrôleur à une partie opérative simulée. En effet, pour les systèmes critiques, des outils commercialisés sont dédiés à la simulation des parties opératives qui, branchées à un contrôleur réel (Hardware In the Loop ou HIL), permettent de faire jouer un contrôleur à valider sans risquer les conséquences d'une faute éventuelle. Ainsi, tout en conservant les interactions entre le contrôleur et la partie opérative, les conséquences d'une non-validité potentielle de la commande sur le système commandé sont évitées.

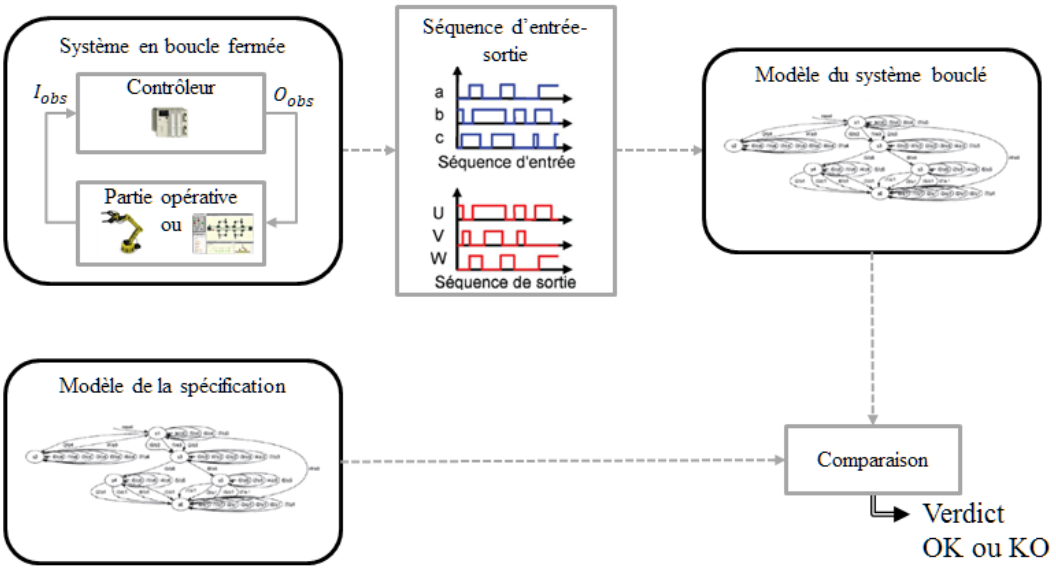


Figure 4.1 – Principe de la validation en boucle fermée

Cependant, le fait de coupler le contrôleur avec sa partie opérative va restreindre les séquences possibles. Ceci s'explique par le fait que le modèle de spécification à partir duquel est conçu le programme de commande est complet. Le programme de commande est donc capable d'émettre une sortie quelque soit les combinaisons des variables d'entrée qu'il reçoit. De son côté, la partie opérative est soumise à des contraintes physiques et technologiques qui ont pour conséquence que certaines combinaisons de variables d'entrée ne seront jamais perçues. Un exemple classique est la présence de deux capteurs de fin de course pour un même actionneur : Ils n'émettront jamais un signal à $Vrai$ simultanément.

Ainsi, le modèle représentant le comportement du contrôleur couplé à sa partie opérative est une restriction du modèle de spécification M_{spec} par le modèle de la partie opérative M_{plant} qui n'est pas connu. La conséquence directe de cette restriction est qu'il est impossible de savoir à priori quel est le nombre de transitions à franchir pour assurer que la totalité du comportement possible du contrôleur, lorsqu'il est couplé à sa partie opérative, est validé. Il est donc nécessaire de mettre en place une stratégie d'observation afin de pouvoir déterminer à partir de quel moment il est considéré que suffisamment de valeurs d'entrée/sortie ont été observées et analysées pour considérer le contrôleur logique comme valide. Cette analyse se fait par comparaison avec le modèle de spécification à travers une relation similaire à la relation de conformité.

Mais avant de développer les solutions mises en oeuvre pour réaliser les diverses

étapes qui permettront de valider le contrôleur logique couplé à sa partie opérative, il convient de mettre en évidence les avantages et les inconvénients de cette méthode par rapport à la méthode usuelle pour valider un contrôleur logique : le test de conformité.

4.1.2 Comparaison de la validation en boucle fermée avec le test de conformité

Le test de conformité, comme il a été vu dans le chapitre précédent, permet de rendre un verdict sur le comportement d'un contrôleur logique isolé qui garantisse la validité de ce comportement selon un objectif défini. Dans le cas traité, le test est capable d'assurer que la séquence de test est passée par toutes les transitions du modèle de spécification et que le contrôleur logique à réagi comme attendu pour chacune de ces transitions.

Dans le cas d'une validation en boucle fermée, une première comparaison rapide avec le test de conformité fait apparaitre plusieurs inconvénients :

- Ce type de validation, par définition, nécessite une partie opérative à coupler au contrôleur. Celle-ci peut être réelle ou simulée et doit être exempte de faute.
- Le modèle du comportement du système en boucle fermée est inconnu. Seul le modèle de spécification, qui contient tout le comportement du système en boucle fermée mais aussi des évolutions impossibles pour ce système en boucle fermée, peut être utilisé pour l'analyse des séquence observées.
- Comme le modèle du comportement du système en boucle fermée n'est pas connu, la durée d'observation pour garantir la validité du contrôleur est elle aussi inconnue.

Cependant, une des difficultés majeures lorsque les techniques de test sont appliquées à des cas industriels est la longueur de la séquence de test. En effet, les modèles formels de spécification atteignent très vite des tailles importantes (voir la Table 2.2 en section 2.4.2) et leur parcours peut entrainer des délais non-acceptables pour les industriels. La restriction du comportement du contrôleur par le couplage avec la partie opérative permet alors de **limiter la taille du modèle à valider** en réduisant l'analyse aux seuls comportements possibles pour la partie opérative.

D'autre part, un processus de validation cherche par définition (Boehm (1979)) à s'assurer que l'on fait le bon produit, c'est-à-dire dans notre cas que le contrôleur contrôle la partie opérative conformément à sa spécification. Coupler le contrôleur à valider avec

sa partie opérative permet donc de se placer dans la configuration d'utilisation de ce contrôleur et par conséquent de fournir un verdict qui soit mieux adapté à la définition de la validation que lorsque ce contrôleur est isolé ou modélisé.

C'est sous cette considération qu'une technique de validation en boucle fermée est présentée dans ce mémoire. Dans un premier temps, une solution est proposée afin de pallier au troisième inconvénient et être capable de donner un critère permettant de mesurer le comportement observable et de déterminer la fin de la phase d'observation. L'analyse des séquences d'entrée/sortie sera ensuite présentée. Ces deux opérations peuvent être toutes deux réalisées en ligne.

4.1.3 Critère de fin d'observation

L'idée de construire un modèle d'un système à partir de l'observation de son comportement est le principe sur lequel se basent les techniques d'identification présentées en section 1.4.2.3. C'est donc à partir des travaux développés dans ce domaine que le critère de fin d'observation est défini. En effet, lors de l'identification en boite noire ou grise, la connaissance de la taille totale du modèle à construire peut aussi être une inconnue.

Klein et al. (2005) place ses travaux dans le cadre du diagnostic et cherche à reconstruire un modèle représentant exactement, à une longueur de séquence donnée, le comportement observé. Dans son cas, chaque pas d'observation est enregistré dans un unique vecteur composé par la concaténation du vecteur représentant la valeur des variables d'entrée et celui représentant la valeur des variables de sortie. Ainsi, il est capable de répertorier toutes les différentes combinaisons observées, et notamment lorsqu'une qui n'a encore jamais été observée arrive. En comptant les occurrences de nouvelles observations on peut alors tracer une courbe de la forme de celle présentée en Figure 4.2.

On peut voir que passé un certain nombre de cycles API, le nombre de nouvelles valeurs d'entrée - sorties jamais encore observées diminue jusqu'à ce que la courbe montre une convergence vers une valeur donnée. Lorsque cette convergence est atteinte, cela signifie que toutes les combinaisons de valeur des variables d'entrée ou de sortie possibles ont été observées au moins une fois et la phase d'observation peut être stoppée.

Dans notre cas, la connaissance du modèle de la spécification est disponible. Hors, chaque observation des valeurs des variables d'entrée et de sortie correspond au franchissement d'une transition du modèle de comportement du programme de commande qui

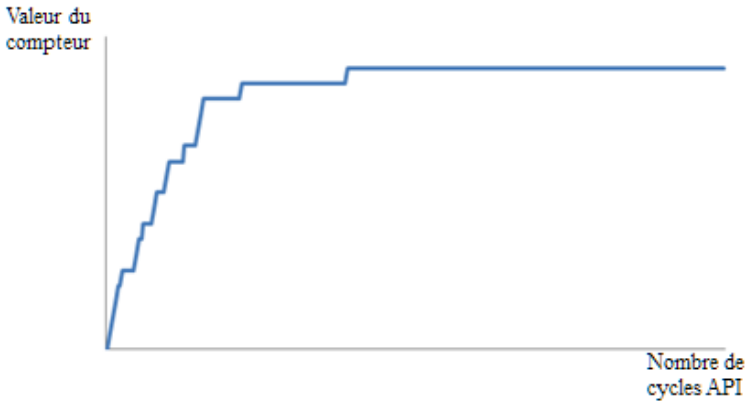


Figure 4.2 – Evolution du nombre d'observations de valeurs d'entrée sortie différentes présentée dans Klein et al. (2005)

doit correspondre à une transition du modèle de spécification si elle est valide. Ainsi, à chaque observation, non seulement la connaissance de la valeur des variables d'entrée et de sortie mais aussi la transition du modèle de spécification correspondante est accessible. On peut alors définir un critère de fin d'observation ainsi :

- A chaque pas d'observation correspondant au franchissement d'une nouvelle transition du modèle de spécification, un compteur est incrémenté ; la valeur courante de ce compteur représente le nombre de transitions qui ont été franchies au moins une fois pendant la durée de l'observation.
- Lorsque la valeur de ce compteur demeure inchangée pendant un temps pré-défini, la phase d'observation se termine.

Le critère de fin d'observation est donc qu'il n'y ait aucun franchissement de transition qui n'ait pas déjà été franchie auparavant pendant un temps suffisant.

Ces résultats concernent la phase d'observation de la validation en boucle fermée. Il convient maintenant de définir la méthode d'analyse des séquences d'entrée/sortie observées.

4.2 Validation lorsque l'observation est réalisée à l'intérieur de l'API

Dans un premier temps, afin de faciliter la mise en place de la méthode, il est supposé que l'observation des entrées et des sorties se fait directement à l'intérieur de l'API, comme illustré en Figure 4.3. Pour cela, il est nécessaire de modifier le programme de

commande pour que celui-ci copie les valeurs des variables d'entrée lues et des variables de sortie émises dans une table des variables. Les valeurs stockées dans cette table sont ensuite transmises à un observateur extérieur via un protocole de communication (ModBus par exemple) qui pourra alors analyser la séquence de valeurs reçue.

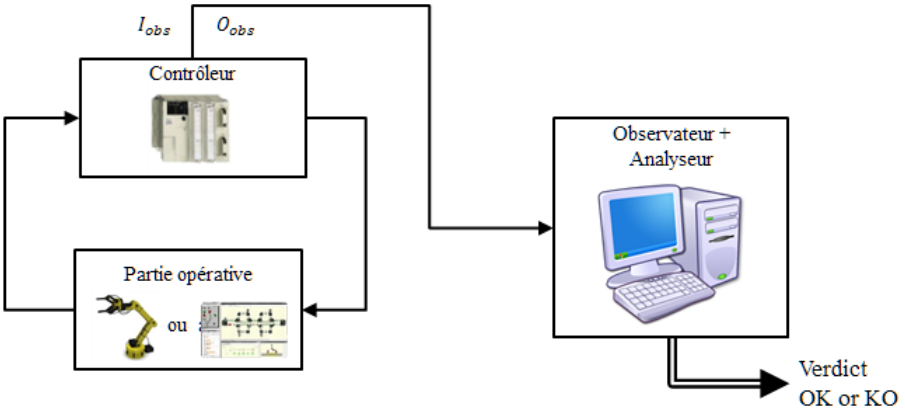


Figure 4.3 – Principe de l'observation à l'intérieur de l'API

Ainsi, les variables d'entrée et de sortie observées sont directement celles lues et envoyées par l'API. Il est alors possible de synchroniser l'API avec l'observateur pour garantir une observation périodique calée sur la lecture et la mise à jour des variables d'entrée et de sortie. Cela a pour principal avantage de supprimer tous les phénomènes inhérents au mode de lecture des entrées de l'API, tels que le retard à l'émission des sorties, la lecture synchrone d'évènements asynchrones ou au contraire la lecture asynchrone d'évènements synchrones. Ce dernier phénomène étant possible par exemple dans le cas où la partie opérative est simulée ou encore si des capteurs sont corrélés.

Cette section présente donc une méthode de construction du modèle du système en boucle fermée et de validation du contrôleur dans ce système en boucle fermée.

4.2.1 Principe de la méthode de construction et de validation du modèle du système en boucle fermé

Dans le cas pris en compte dans ces travaux, la validation en boucle fermée requiert la construction d'un automate à état fini représentant le modèle du système en boucle fermée sous la forme d'une machine de Mealy afin de pouvoir effectuer la comparaison avec le modèle de la spécification.

De plus, le comportement possible du système en boucle fermée est une restriction du comportement du programme de commande lorsque le contrôleur est isolé. Le modèle du

comportement du programme de commande sans faute étant le modèle de spécification, il est intéressant d'utiliser cette connaissance pour construire le modèle du système en boucle fermée.

Ainsi, une technique d'identification en boite grise d'une machine de Mealy est nécessaire. Aucune technique adaptée n'ayant pas encore été développé pour ce cas, une nouvelle technique sera proposée dans ce mémoire.

En effet, si le modèle de spécification est sous la forme d'une machine de Mealy, la partie opérative et le programme de commande peuvent alors aussi voir leur comportement modélisées via une machine de Mealy qui partage les même alphabets I_{spec} et O_{spec}. Il est à noter que là où l'alphabet d'entrée est I_{spec} et l'alphabet de sortie est O_{spec} pour le modèle de comportement du programme de commande, l'alphabet d'entrée est O_{spec} et l'alphabet de sortie est I_{spec} pour la partie opérative. De plus, ces deux modèles sont synchronisés et les évolutions de chacun d'entre eux sont contraintes par l'autre modèle.

Ainsi, en considérant que le langage accepté par chacun des modèles est composé de mots sous la forme de couples (I_{spec}, O_{spec}) on note L_p le langage accepté par le modèle de la partie opérative et L_{impl} le langage accepté par le modèle de comportement du programme de commande lorsque le contrôleur est isolé. Le langage accepté par le modèle du système en boucle fermée L_{boucle} est alors l'intersection de ces deux langages, comme illustré par la Figure 4.4. Dans le cas où l'implantation est conforme à la spécification, les modèles de comportement du programme de commande et de la spécification sont identiques. On peut alors dire que le langage du système en boucle fermée L_{boucle} est inclus dans le langage de la spécification. Autrement dit, cela signifie que le modèle du système en boucle fermée est composé d'un sous-ensemble du modèle de spécification qui correspond aux valeurs d'entrée et de sortie observées.

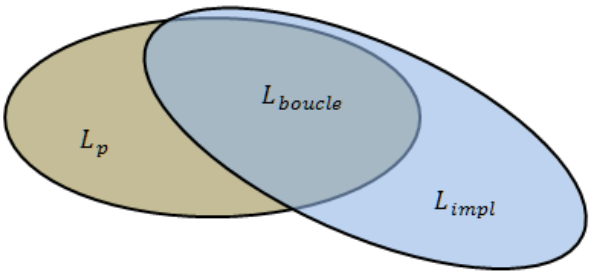


Figure 4.4 – Langage du modèle du système en boucle fermée

Ainsi, chaque couple de valeurs d'entrée et de sortie peut être interprété comme l'étiquette associée à une transition du modèle de spécification. L'état initial du système étant connu et le modèle de spécification déterministe, il est possible de faire évoluer le modèle de spécification selon les entrées et sorties observées.

Hors, chaque transition du modèle de spécification ainsi franchie correspond à une évolution du système en boucle fermée et appartient donc au modèle du système en boucle fermée. Il est donc possible de construire le modèle du système en boucle fermée en dupliquant pas à pas les transitions franchies dans le modèle de spécification.

On obtient alors un modèle dont le langage est un sous-ensemble du modèle de spécification et qui inclus la séquence d'observation. Il faut cependant remarquer que chaque couple d'entrée/sortie est associé à une transition existante partant de l'état courant du modèle de spécification. Si le couple observé ne correspond à aucune transition existante, alors cela traduit un comportement non valide.

Les phases de construction et de validation du modèle du système en boucle fermée sont donc réalisées en même temps.

4.2.2 Algorithme de construction du modèle du système en boucle fermée

Cette section présente l'algorithme de construction et de validation du modèle du système en boucle fermée. Ce modèle est une machine de Mealy définie par un 6-uplet (I_{bf}, O_{bf}, S_{bf}, s_{initbf}, δ_{bf}, λ_{bf}). Comme ce modèle décrit un langage qui est inclus dans la spécification du contrôleur, leurs alphabets sont identiques. De même, l'état initial du modèle en boucle fermée est le même que celui isolé. Ces trois éléments sont aussi identiques dans le modèle de spécification. De plus, le modèle du système en boucle fermée construit étant celui conforme à la spécification, l'ensemble des états comme les fonctions de transition et de sortie sont des sous-ensembles des éléments correspondants pour le modèle de la spécification. On peut donc définir les éléments du modèle du système en boucle fermée ainsi :

- $I_{bf} = I_{spec}$
- $O_{bf} = O_{spec}$
- $S_{bf} \subseteq S_{spec}$

- $s_{initbf} = s_{init}$
- $\delta_{bf} \subseteq \delta_{spec}$
- $\lambda_{bf} \subseteq \lambda_{spec}$

La construction de ce modèle à partir d'une séquence de valeurs d'entrée et de sortie observée est décrite à travers l'algorithme 5.

Algorithme 5 Construction du modèle du système en boucle fermée

1: Soit $\sigma = (IO_1, ..., IO_n)$ la séquence observée avec $IO_k = (i_k, o_k)$ où $i_k \in I_{spec}$ et $o_k \in O_{spec}$
2: Soit $s = s_{initbf}$ l'état courant du modèle en construction ; $S_{bf} = s_{initbf}$
3: **for** $IO = (i, o) \in \sigma$ **do**
4: **if** il existe $s_d \in S_{spec}$ tel que $\delta_{spec}(s, i) = s_d$ et $\lambda_{spec}(s, i) = o$ **then**
5: **if** $s_d \notin S_{bf}$ **then**
6: $S_{bf} = S_{bf} \cup s_d$
7: **end if**
8: **if** le franchissement de la transition $\delta_{bf}(s, i)$ n'a jamais encore été observé **then**
9: $\delta_{bf}(s, i) = s_d$
10: $\lambda_{bf}(s, i) = o$
11: **end if**
12: $s = s_d$
13: **else**
14: L'implantation n'est pas conforme à la spécification
15: **end if**
16: **end for**

Seul l'état initial est connu au début du processus et il n'y a aucun élément dans la fonction de transition δ_{bf} ainsi que dans la fonction de sortie λ_{bf}. Pour chaque pas d'observation des entrées et des sorties, l'algorithme vérifie qu'il existe bien une transition étiquetée par les valeurs observées partant de l'état courant du modèle de spécification. Si cette transition existe et n'a jamais encore été franchie lors des pas d'observation précédents, alors l'algorithme crée cette transition ainsi que l'état aval dans le modèle du système en boucle fermée (lignes 4 à 12). L'analyse se poursuit ensuite depuis cet état aval. Dans le cas où les valeurs des entrées et sorties observées ne peuvent pas être interprétées comme le franchissement d'une des transition du modèle de spécification, alors la construction de l'automate est interrompue et le contrôleur est déclaré non valide (ligne 14).

On peut noter que cet algorithme peut être exécuté en ligne. Cela signifie que la construction est réalisée et un verdict est rendu à chaque pas d'observation ; la phase

d'observation peut alors être interrompue dès qu'une évolution non-valide est détectée.

4.2.3 Exemple

La technique de construction et de validation du modèle du système en boucle fermée est illustrée dans cette section à travers un exemple simple. Cet exemple représente le contrôle d'un portail automatisé. Il possède 4 entrées $V_I = \{P_o, P_f, Voit, Tel\}$ et 2 sorties $V_O = \{Ouv, Fer\}$. Le modèle de spécification est présenté sous la forme d'un Grafcet en Figure 4.5 et sous la forme d'une machine de Mealy en Figure 4.6. Par souci de visibilité, un seul arc représente toutes les transitions entre deux états.

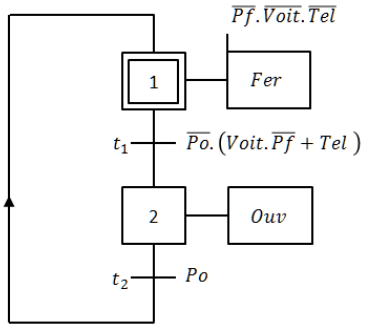


Figure 4.5 – Modèle Grafcet de la spécification

Cette machine de Mealy est composée de 3 états et de 48 transitions ce qui correspond aux 2^4 transitions pour chaque état requises afin d'assurer la complétude.

4.2.3.1 Construction du modèle du système en boucle fermée

Le contrôleur exécutant le programme de commande à valider est couplé à sa partie opérative et les valeurs des entrées et sorties sont observées. La Figure 4.7 montre le chronogramme correspondant au début de la séquence observée.

Chaque pas d'observation est alors traité afin de reconstruire le modèle du système en boucle fermée. Pour cela, l'algorithme recherche pour chaque couple d'entrée/sortie observé quel est le franchissement de transition de la machine de Mealy correspondante dans le modèle de spécification. Un couple d'entrée sortie s'écrit sous la forme $IO = ((P_o, P_f, Voit, Tel), (Ouv, Fer))$. Le premier pas d'observation, selon le chronogramme présenté en Figure 4.7, vaut alors $IO_1 = ((Faux, Vrai, Faux, Faux), ((Faux, Faux))$. Cela correspond à la combinaison de variable d'entrée $i_{spec} = [e]$ et à la combinaison de valeur des variables de sortie $o_{spec} = [A]$ dans le modèle de spécification. Depuis

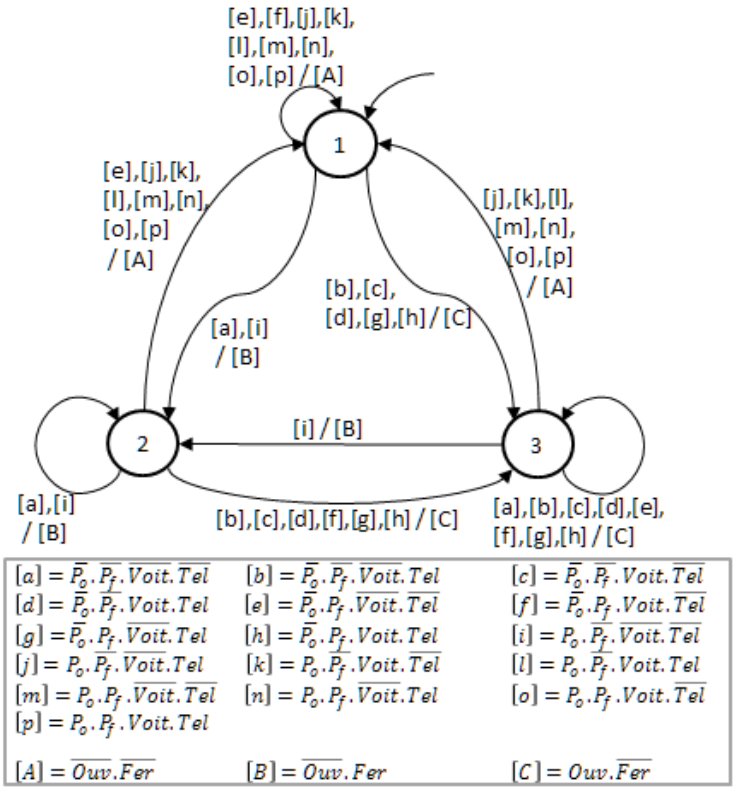


Figure 4.6 – Modèle formel de la spécification

l'état initial qui est l'état 1 de la machine de Mealy, ce couple de valeurs d'entrée et de sortie correspond au franchissement de la self-loop sur cet état étiqueté par $[e]/[A]$. En effet, les fonctions de transition et de sortie valent respectivement $\delta_{spec}(1,[e]) = 1$ et $\lambda_{spec}(1,[e]) = [A]$.

Le franchissement d'une transition s'écrit ainsi :

$$t_{franchie} = \begin{pmatrix} s_u \\ s_d \\ i_{spec} \end{pmatrix}$$

où s_u, s_d et i_{spec} sont respectivement l'état amont, l'état aval et la combinaison de valeurs des variables d'entrée de la transition franchie. La séquence de franchissement de transition correspondant aux différents couples d'entrée sortie observés lors de la phase d'observation décrite dans la Figure 4.7 est alors :

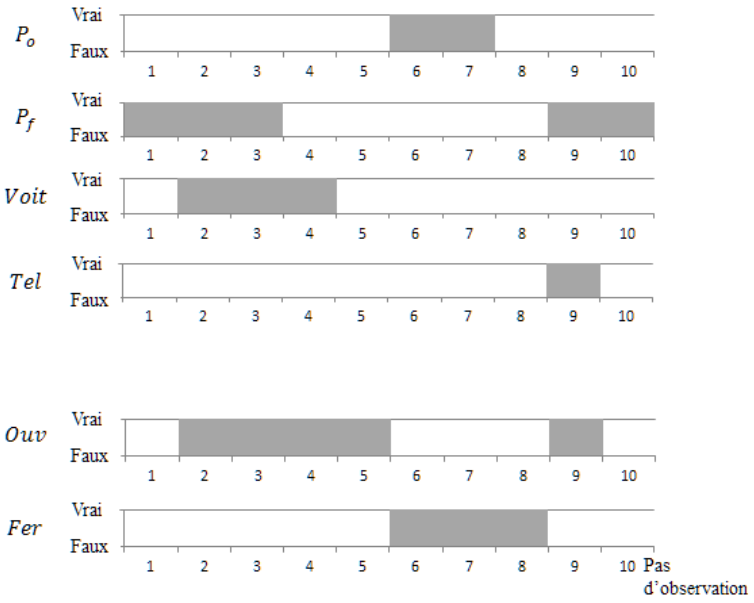


Figure 4.7 – Chronogramme des valeurs des variables d'entrée et de sortie observées

$$T_{franchies} = \left(\begin{pmatrix} 1 \\ 1 \\ [e] \end{pmatrix}, \begin{pmatrix} 1 \\ 3 \\ [g] \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ [g] \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ [c] \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ [a] \end{pmatrix}, \begin{pmatrix} 3 \\ 2 \\ [i] \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ [i] \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ [a] \end{pmatrix}, \begin{pmatrix} 2 \\ 3 \\ [f] \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ [e] \end{pmatrix}, \ldots \right) \tag{4.1}$$

A partir de la séquence de franchissement des transitions du modèle de spécification, il ne reste plus qu'à reconstruire le modèle du système en boucle fermée en copiant les transitions franchies ainsi que les états atteints. La Figure 4.8 représente la structure obtenue pour le modèle du système en boucle fermée à partir de la séquence d'entrée sortie observée.

Afin de définir la longueur de la phase d'observation, un compteur est incrémenté à chaque nouvelle transition du modèle de spécification franchie. La Figure 4.9 montre l'évolution de ce compteur en fonction du nombre de cycle API passés. Cette courbe converge vers une valeur de 36 transitions différentes franchies. Cela signifie que, pour une durée donnée, aucune nouvelle transition n'est franchie après que cette valeur ait été atteinte.

Le modèle du système en boucle fermée final est présenté Figure 4.10. Ce modèle

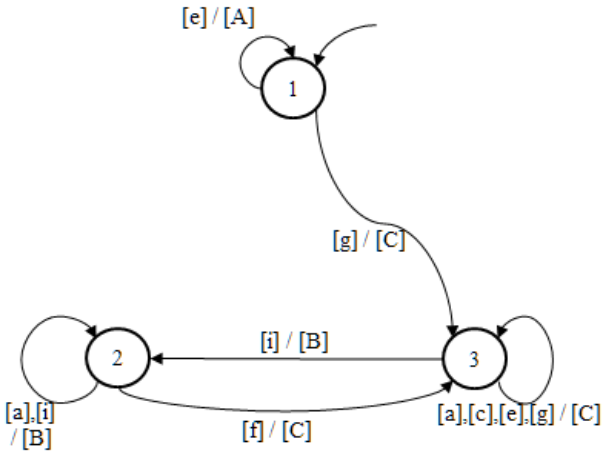


Figure 4.8 – Modèle partiel du système en boucle fermée construit à partir de la séquence observée représentée en Figure 4.7

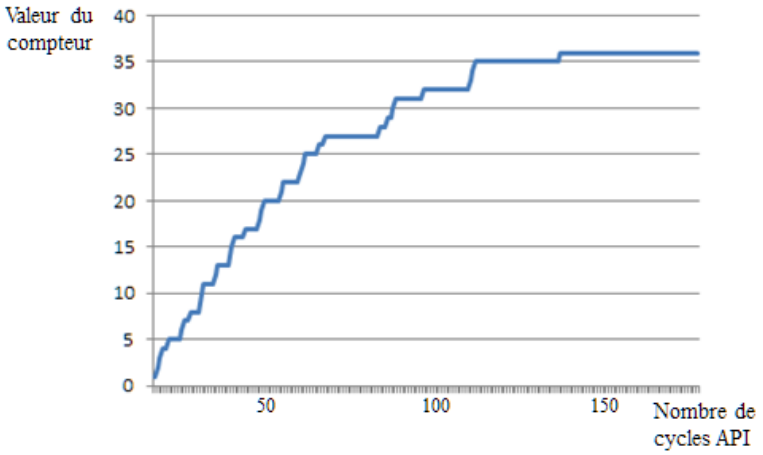


Figure 4.9 – Nombre de franchissements de transitions lors de la phase d'observation

contient 3 états et 36 transitions correspondant aux 36 transitions différentes comptées dans la figure précédente. Cela signifie que 12 transitions du modèle de spécification qui correspondent à toutes les transitions dont la combinaison de valeur de variables d'entrée contient les variables P_o et P_f toutes deux à la valeur $Vrai$. Cette combinaison n'est en effet pas possibles lorsque le contrôleur est couplé à la partie opérative.

4.2.3.2 Détection d'implantation non-conforme à la spécification

Dans la partie précédente, la séquence observée traduisait un comportement conforme à la spécification. Cette partie va maintenant s'intéresser à la capacité de détection d'un contrôleur logique non valide lors de la construction du modèle du système en boucle fermée. Pour cela, il est supposé que depuis l'état initial les variations de valeur d'entrée et de sortie observées sont celles présentées dans le chronogramme en Figure 4.11.

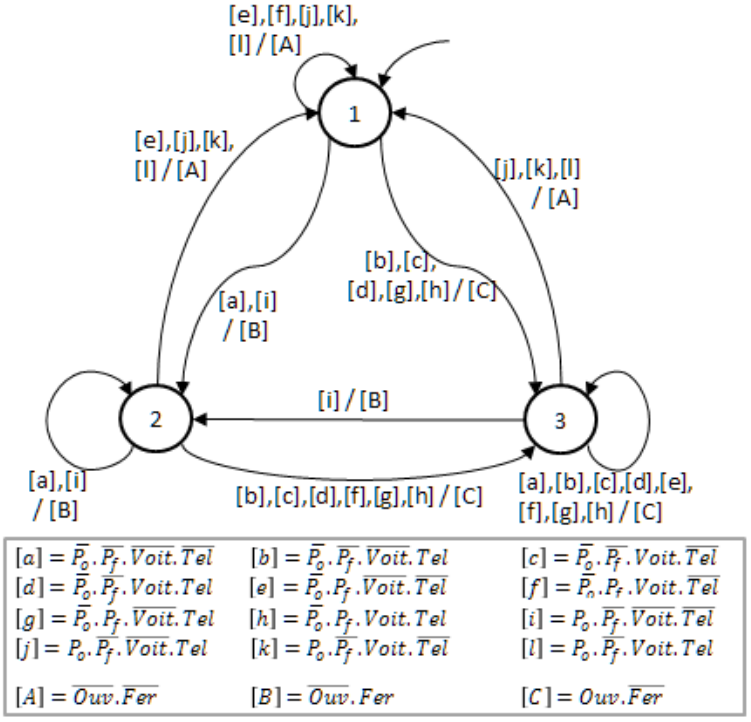


Figure 4.10 – Modèle du système en boucle fermée

Les 4 premiers pas de la séquence d'observation sont identiques à ceux de la Figure 4.7 et seul le 5^{eme} est différent. Lors de l'exécution de l'algorithme 5, les 4 premiers pas vont être correctement associés à leur transition correspondante du modèle de spécification. Le franchissement successif de ces transitions mène à l'état 3 du modèle de spécification. Cependant, le pas d'observation suivant relève une combinaison des valeurs d'entrée valant $[a]$ et une combinaison de valeurs des variables de sortie valant $[A]$. Depuis l'état 3, aucune transition étiquetée $[a]/[A]$ n'existe dans le modèle de spécification. La séquence de transitions franchies devient alors :

$$T_{franchies} = \left(\begin{pmatrix} 1 \\ 1 \\ [e] \end{pmatrix}, \begin{pmatrix} 1 \\ 3 \\ [g] \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ [g] \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ [c] \end{pmatrix}, \emptyset \right) \tag{4.2}$$

Comme l'absence de transition correspondante aux valeurs observées signifie que le comportement observé diffère de celui accepté par la spécification, le contrôleur est déclaré non conforme à la spécification.

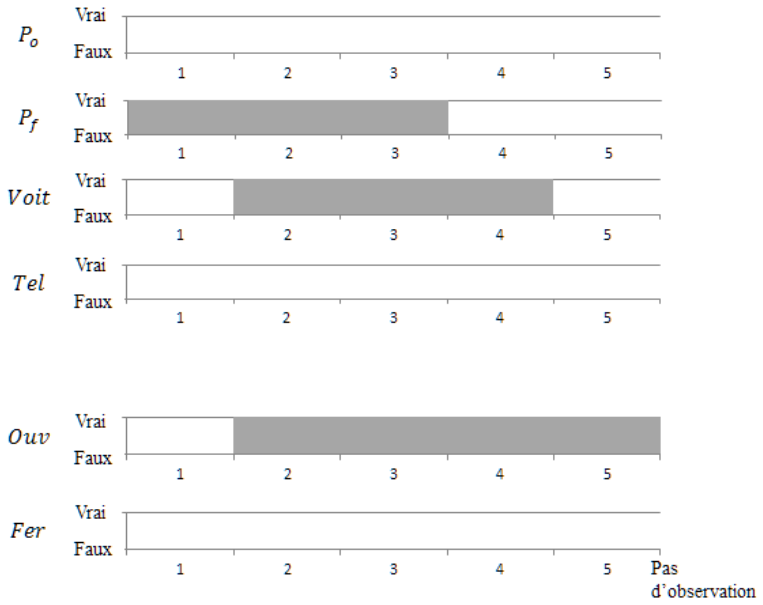


Figure 4.11 – Chronogramme des valeurs des variables d'entrée et de sortie observées

4.2.4 Expérimentation

Cette partie propose l'étude d'un second cas basé sur des relevés expérimentaux. Ceux-ci ont été réalisée à partir d'une plate-forme expérimentale présente au LURPA appelée *Mechatronics Standard System* (MSS) développée par Bosch et présentée en Figure 4.12.

Cette machine traite des roues dentées de divers matériaux (acier, bronze, PVC) en posant et/ou en retirant des bagues. Elle se décompose en quatre postes qui prennent successivement en charge chaque roue dentée :

- Le poste 1 est en charge de la distribution des roues dentées à traiter.
- Le poste 2 est en charge de déterminer la présence ou non de bague ainsi que la matière de la roue dentée.
- Le poste 3 effectue les diverses opérations de pose et de retrait des bagues.
- Le poste 4 trie et range les roues dentées traitées pour évacuation.

L'observation des variables d'entrée et de sortie est réalisée à l'intérieur de l'API. En effet, le programme de commande implanté dans le contrôleur logique copie et stocke les valeurs des variables d'entrée et de sortie à chaque cycle API. Il construit ainsi une séquence de valeurs d'entrée/sortie qu'il incrémente à chaque fin de cycle API.

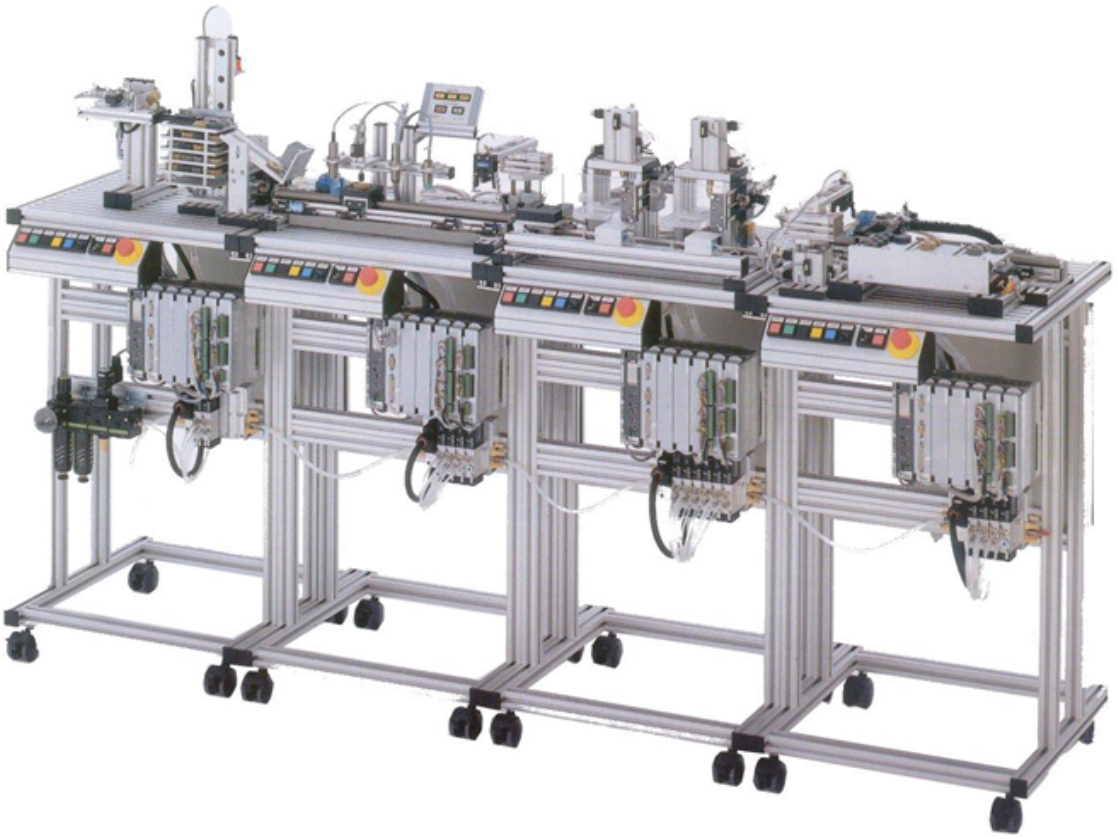


Figure 4.12 – plate-forme expérimentale MSS

Un observateur, sous la forme d'un code Python, exécute alors un script qui initie une communication avec l'API, via une carte réseau, afin de récupérer la séquence de valeurs d'entrée/sortie construite depuis la dernière communication. Après traitement de la séquence reçu, le script attend 100ms avant de se relancer afin d'éviter toute saturation du réseau.

Le système complet comprend 144 variables d'entrée et 80 variables de sortie. Il contient aussi de nombreuses de temporisations utilisées pour dans chacune des stations. Cependant, les modèles développés dans ces travaux font l'hypothèse de l'absence d'éléments temporisés qui se traduisent alors par la présence d'indéterminismes. Une solution à ce problème d'indéterminisme serait d'introduire des variables annonçant la fin de chaque temporisation mais dans un premier temps l'étude d'un sous-système ne contenant pas de temporisation est proposée. On se limite alors au fonctionnement du système de préhension qui déplace la roue en bout de station 3 vers la station 4 montré en Figure 4.13

Cette opération nécessite les informations de 7 capteurs (S3_car-right, grip_closed,

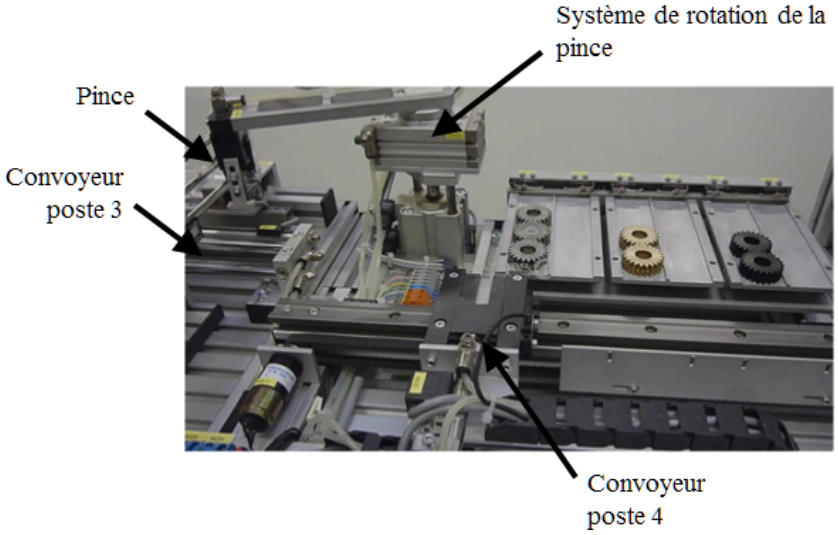


Figure 4.13 – Système de préhension du poste 4

grip_right, grip_left, grip_bottom, grip_top, S4_car_left) et d'agir sur 3 actionneurs (Grip_close, Grip_rot, Grip_down) qui commandent des vérins monophasés. Au repos, la pince est ouverte, en position haute et en butée à droite. La Figure 4.14 présente le modèle de spécification Grafcet.

Le modèle formel de cette spécification contient 9 états et $9 * 2^7 = 1156$ transitions.

Suite à une campagne d'observation, une séquence d'observation d'une longueur de 890 pas a été enregistrée. Les entrées et sorties considérées ont été extraites de cette séquence et ont été analysées selon l'algorithme 5 implanté en Python. Cette analyse a rendu un verdict de validité de l'implantation avec une courbe du nombre de transitions franchies présentée en Figure 4.15. On peut ainsi voir qu'après 250 pas d'observation, le nombre de transitions différentes franchies converge vers une valeur de 22 transitions différentes. La machine de Mealy représentant le comportement du système en boucle fermée contient très peu de transitions par rapport au modèle de spécification, ce qui signifie qu'une grande partie du comportement spécifié (soit 1132 transitions) ne sera jamais atteignable ; une erreur dans le programme de commande conduisant à un mauvais comportement dans cet espace non-atteignable n'aura donc aucune conséquence sur le comportement du système en boucle fermée.

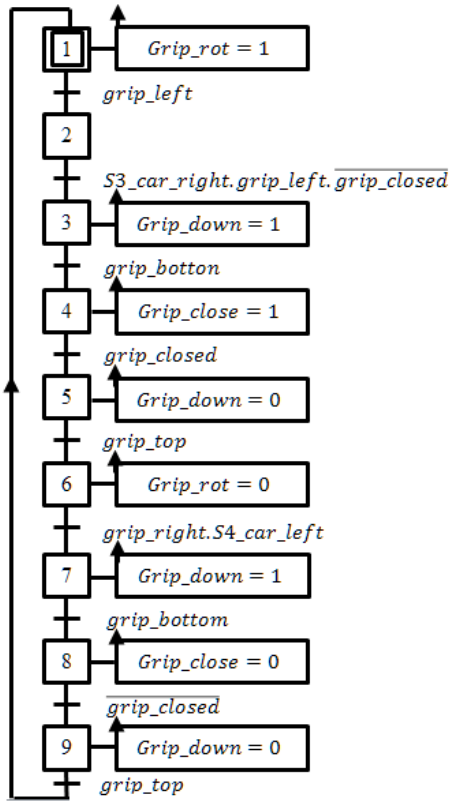


Figure 4.14 – Modèle de spécification du système de préhension

4.3 Validation lorsque l'observation est réalisée aux bornes de l'API

Une première méthode de construction d'algorithme a été proposée dans la section précédente. Cette méthode suppose néanmoins que l'observation des valeurs des variables d'entrée et de sortie est directement effectuée à l'intérieur de l'API. Cela requiert alors de modifier le programme de commande afin que celui-ci envoie une copie des valeurs lues et émises à l'observateur. Le contrôleur logique analysé ne sera donc pas exactement tel qu'il serait en configuration réelle d'utilisation.

Comme l'objectif de ces travaux est justement de se placer dans une configuration qui soit celle d'utilisation, cette section propose d'étendre la technique développée précédemment en considérant que l'observation se fasse cette fois-ci aux bornes de l'API comme illustré par la Figure 4.16. La principale difficulté de cette extension est que l'observateur perd la connaissance de ce que l'API lit réellement. Cela signifie que les phénomènes classiques de retard à l'émission des sorties, lecture synchrone d'évènements asynchrones et lecture asynchrone d'évènements synchrones vont apparaitre lors de la phase d'observation. Il est donc nécessaire d'adapter l'algorithme de construction du mo-

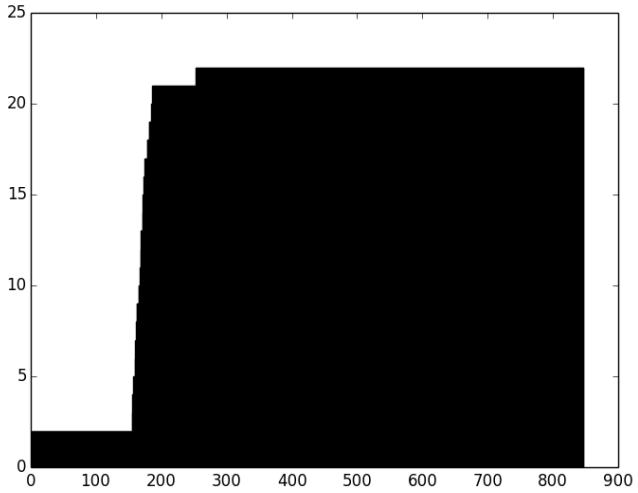


Figure 4.15 – Nombre de transitions franchies durant la phase d'observation de la MSS

dèle du système en boucle fermée afin de le rendre tolérant à ces phénomènes. Pour cela, l'algorithme sera enrichi de deux parties distinctes, l'une qui traite des phénomènes de retard et de lecture asynchrone d'évènements synchrones, similaire à ce que nous avons proposé pour le test de conformité, et une autre partie qui traite des phénomènes de lecture synchrone d'évènements asynchrones.

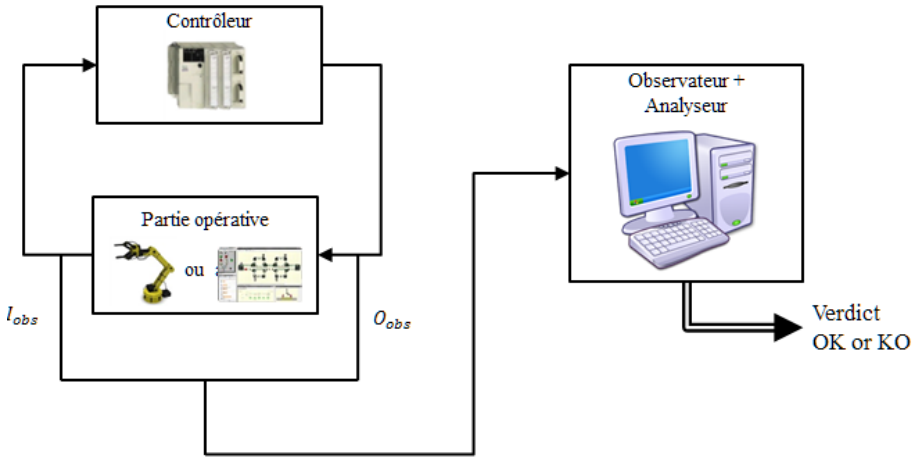


Figure 4.16 – Principe de l'observation aux bornes de l'API

4.3.1 Prise en compte des phénomènes de retard et de lecture asynchrone d'évènements synchrones

Le principe de cette extension de l'algorithme de construction du modèle du système en boucle fermée est le même que lors de la définition de la relation de conformité

présentée en section 3.3. En effet, la construction du modèle se base sur la recherche dans le modèle de spécification de la transition correspondant à la combinaison de valeurs des variables d'entrée et de sortie observée ; si une transition existante dans le modèle de spécification partant de l'état courant est trouvée, alors celle-ci est reportée dans le modèle du système en boucle fermée. Or, la phase de recherche et de sélection de la transition dans le modèle de spécification est identique au processus de test réalisé par la relation de conformité.

Ainsi, si le couple d'entrée/sortie observé ne correspond à aucune transition du modèle de spécification partant de l'état courant, il faudra attendre les couples d'entrée/sortie suivants afin de chercher à déterminer si le comportement observé décrit une non-validité ou une mauvaise lecture des évènements d'entrée. On rappelle que le cas d'une lecture asynchrone d'évènements synchrones se traduit dans le modèle de spécification par un franchissement successif de transitions représentant :

- un éventuel premier pas d'observation traduisant le décalage d'un cycle API de la réponse aux nouvelles valeurs des variables d'entrée,
- le franchissement d'une première transition étiquetée par une combinaison de variables d'entrée qui soit une mise à jour partielle des valeurs d'entrée entre les deux combinaisons de valeurs observées,
- le franchissement d'une seconde transition étiquetée par la combinaison de valeurs des variables d'entrée correspondant à celle observée.

Par contre, tout comme pour la relation de conformité, ce test ne cherche qu'à détecter un éventuel phénomène de retard ou de désynchronisation. Par manque d'information sur les valeurs des variables d'entrée lues, il n'est pas possible d'être certain des transitions effectivement franchies. Seul le nouvel état atteint, caractérisé par la valeur des variables de sortie, peut être garanti. Ainsi, si le test sur la désynchronisation est concluant, aucune transition ne sera reportée dans le modèle de spécification et seul l'état final atteint après les trois pas d'observation sera construit.

La construction du modèle du système en boucle fermée est réalisée par l'algorithme 6 qui se décompose ainsi :

- Initialisation des variables (lignes 1 à 3),

Algorithme 6 Construction du modèle du système en boucle fermée prenant en compte les phénomènes de désynchronisation et de retard

1: Soit $\sigma = (IO_1, ..., IO_n)$ la séquence observée avec $IO_k = (i_k, o_k)$ où $i_k \in I_{spec}$ et $o_k \in O_{spec}$
2: Soit $s = s_{initbf}$ l'état courant du modèle en construction ; $S_{bf} = s_{initbf}$
3: $k = 1$ Initialisation du compteur
4: **while** $k \leq n$ **do**
5: **if** il existe $s_d \in S_{spec}$ tel que $\delta_{spec}(s, i_k) = s_d$ et $\lambda_{spec}(s, i_k) = o_k$ **then**
6: **if** $s_d \notin S_{bf}$ **then**
7: $S_{bf} = S_{bf} \cup s_d$
8: **end if**
9: **if** le franchissement de la transition $\delta_{bf}(s, i_k)$ n'a jamais encore été observé **then**
10: $\delta_{bf}(s, i_k) = s_d$
11: $\lambda_{bf}(s, i_k) = o_k$
12: **end if**
13: $s = s_d$
14: $k = k + 1$
15: Retour à la ligne 4.
16: **else**
17: **if** $\lambda_{spec}(s, i_{k-1}) = o_k$ **then**
18: $s_d = \delta(s, i_{k-1})$
19: $s = s_d$
20: $k = k + 1$
21: **end if**
22: **if** $\exists I^x \in I_M$ tel que $(I^x \backslash I^i \cup I^i \backslash I^x) \subseteq (I^i \backslash I^j \cup I^j \backslash I^i)$ et $\lambda_{spec}(s_c, I^x) = O^k$ **then**
23: **if** $i_k = i_{k+1}$ et $\exists s_d \in S_M$ qui vérifie $\delta_{spec}(s, I^x) = s_d$ et $\lambda_{spec}(s_d, i_k) = o_{k+1}$ **then**
24: $k = k + 2$
25: $s = \delta_{spec}(s_d, i_k)$
26: Retour à la ligne 4.
27: **else**
28: L'implantation n'est pas conforme à la spécification
29: **end if**
30: **else**
31: L'implantation n'est pas conforme à la spécification
32: **end if**
33: **end if**
34: **end while**

- Construction du modèle si les valeurs observées sont cohérentes avec le modèle de spécification (lignes 5 à 15),
- Si non, analyse de la suite des valeurs de variables observées pour rechercher un phénomène dû au cycle de lecture de l'API (à partir de la ligne 16),
- Recherche d'un éventuel phénomène de retard (lignes 17 à 21),
- Recherche sur les deux pas d'observation suivants si les valeurs observées correspondent à un phénomène de lecture asynchrone d'évènements synchrones ; on remarque qu'un retard seul est aussi accepté par cette analyse (lignes 22 à 25),
- Verdict de non validité de l'implantation du programme de commande si les divers test sont non-concluants (lignes 26 à 31).

On peut remarquer que cet algorithme fait l'hypothèse qu'après un changement de valeur synchrone de plusieurs variables d'entrée étant perçu comme asynchrone, il n'y aura aucun nouveau changement de valeur des variables d'entrée pendant les deux cycles API suivants. Cette hypothèse est limitante, mais sans information sur les valeurs des variables d'entrée effectivement lues par l'API, elle est nécessaire. En effet, la prise en compte d'un évènement d'entrée pendant un phénomène de lecture asynchrone d'évènements synchrones augmenterait de 2^n, avec n le nombre de variables d'entrée, le nombre de franchissement de transitions acceptées. Cela nuirait alors au verdict de non-validité qui serait fortement limité.

Reste maintenant à prendre en compte le cas de lecture synchrone d'évènements asynchrones, aussi appelé phénomène de synchronisation. Ce cas ne pouvait pas se produire lors de l'exécution d'un test de conformité, puisque les changements de valeurs des variables d'entrée sont espacés d'au moins 3 cycles API. La sous-section suivante propose donc d'étudier les conséquences de ce phénomène et une extension de l'algorithme de construction du modèle du système en boucle fermée afin de le prendre en compte.

4.3.2 Prise en compte du phénomène de synchronisation

Comme défini dans la section 1.1.3, le phénomène de lecture synchrone d'évènements asynchrones est un phénomène qui est courant pour les API. Comme l'observateur et

l'API ne sont pas synchronisés, un décalage entre leur lecture des entrées respectives peut faire apparaitre ce phénomène, comme illustré en Figure 4.17.

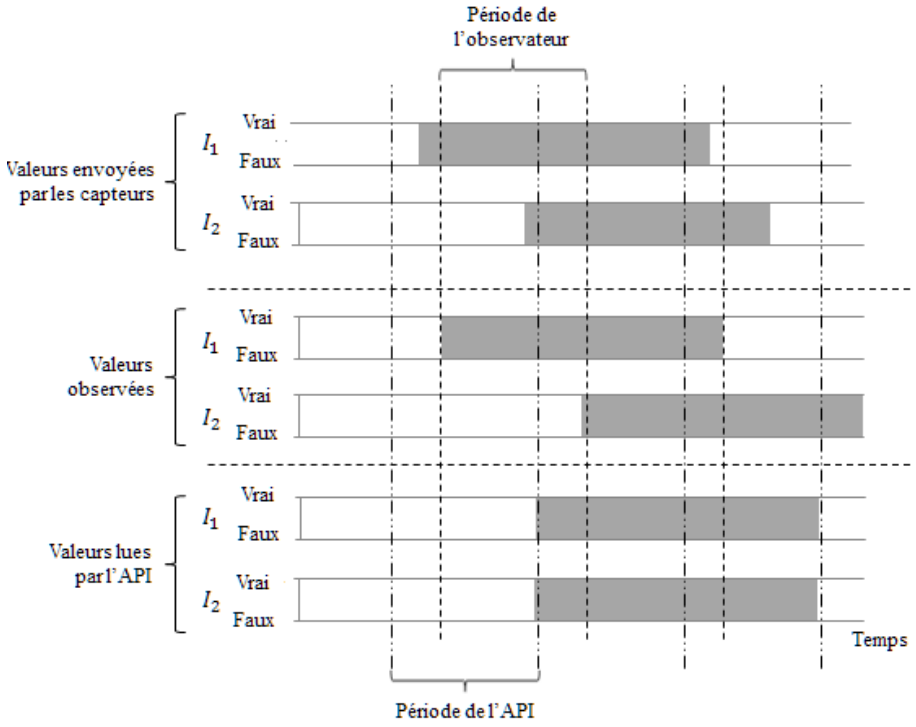


Figure 4.17 – Chronogramme montrant le décalage temporel entre les variables d'entrée émises par la partie opérative, celles observées et celles lues par l'API

Ainsi, en considérant la machine de Mealy générique partielle présentée en Figure 4.18 et en supposant que l'état s_1 est actif avec pour couple d'entrée sortie observé (I^i, O^i), la lecture synchrone d'évènements asynchrones I^j puis I^t se traduit ainsi :

- Au premier pas, l'API n'a pas encore perçu le changement de valeur des variables d'entrée I^j. Il émet donc la même valeur des variables de sortie O^i ce qui correspond au franchissement d'une self-loop sur l'état s_1 portant la valeur initiale des variables d'entrée I^i.

- Au second pas, l'API perçoit la bonne nouvelle valeur des variables d'entrée I^t, constituée des deux mises à jour successives observées. Il émettra donc comme valeur des variables de sortie O^l, correspondant au franchissement de la transition vers s_4 étiquetée par la valeur des variables d'entrée I^t observée au second pas depuis l'état actif s_1.

D'une manière générale, une lecture synchrone d'évènements asynchrones se traduit dans le modèle de comportement du programme de commande par :

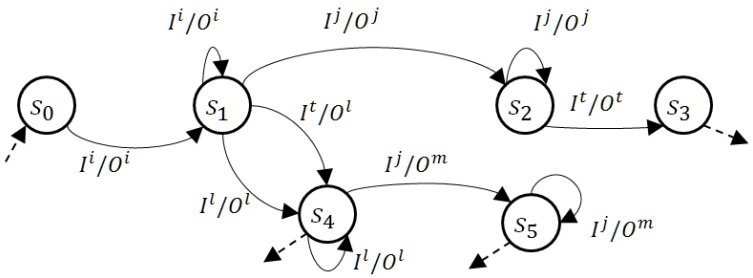


Figure 4.18 – Machine de Mealy générique partielle déjà présentée en Figure 3.7

- Au premier cycle API, le franchissement de la transition étiquetée par la même valeur des variables d'entrée que le cycle API précédent.

- Au second cycle API, le franchissement de la transition étiquetée par la nouvelle valeur des variables d'entrée.

Au sein de l'algorithme, cela peut être rapproché du phénomène de retard pendant lequel a lieu un changement de valeur des variables d'entrée. Le nouvel algorithme 7 permet la construction du modèle du système en boucle fermée tout en étant robuste aux divers phénomènes dus à la lecture cyclique des valeurs des variables d'entrée de l'API. Ce dernier algorithme est issu du précédent en ajoutant la prise en compte de la possibilité d'un phénomène de synchronisation lorsqu'un retard sur la mise à jour des sorties est détecté (lignes 17 à 22).

Une nouvelle fois, il est supposé ici qu'un seul phénomène de synchronisation ou de désynchronisation peut apparaitre à la fois. Le but de cette hypothèse étant de ne pas trop étendre le nombre de transitions sur lesquelles aucun verdict de validité n'est donné afin de ne pas rater une non-validité. Cependant, le cas où, depuis un état du modèle du système en boucle fermée, une combinaison de variable d'entrée particulière est toujours associée à une valeur des variables de sortie qui ne soit pas celle attendue par la spécification mais qui soit impliquée dans un phénomène de synchronisation ou de désynchronisation est possible. Dans ce cas, il peut être nécessaire de tester cette transition précise via un test de conformité afin de lever l'incertitude.

La suite de cette section présente un exemple de construction du modèle du système en boucle fermée lorsque l'observation est réalisée au bornes de l'API et comment l'algorithme réagit à des phénomènes de synchronisation et de désynchronisation des évènements d'entrée.

Algorithme 7 Construction du modèle du système en boucle fermée prenant en compte les phénomènes dûs à l'API

1: Soit $\sigma = (IO_1, ..., IO_z)$ la séquence observée de longueur z avec $IO_k = (i_k, o_k)$ où $i_k \in I_{spec}$ et $o_k \in O_{spec}$
2: Soit $s = s_{initbf}$ l'état courant du modèle en construction ; $S_{bf} = s_{initbf}$
3: $k = 1$ Initialisation du compteur
4: **while** $k \leq z$ **do**
5: **if** il existe $s_d \in S_{spec}$ tel que $\delta_{spec}(s, i_k) = s_d$ et $\lambda_{spec}(s, i_k) = o_k$ **then**
6: **if** $s_d \notin S_{bf}$ **then**
7: $S_{bf} = S_{bf} \cup s_d$
8: **end if**
9: **if** le franchissement de la transition $\delta_{bf}(s, i_k)$ n'a jamais encore été observé **then**
10: $\delta_{bf}(s, i_k) = s_d$
11: $\lambda_{bf}(s, i_k) = o_k$
12: **end if**
13: $s = s_d$
14: $k = k + 1$
15: Retour à la ligne 4.
16: **else**
17: **if** $\lambda_{spec}(s, i_{k-1}) = o_k$ **then**
18: $s_d = \delta(s, i_{k-1})$
19: **if** $i_k \neq i_{k+1}$ et $\lambda(s_d, i_{k+1}) = o_{k+1}$ **then**
20: $s = \delta(s_d, i_{k+1})$
21: $k = k + 2$
22: Retour à la ligne 4.
23: **else**
24: $s = s_d$
25: $k = k + 1$
26: **end if**
27: **end if**
28: **if** $\exists I^x \in I_M$ tel que $(I^x \backslash I^i \cup I^i \backslash I^x) \subseteq (I^i \backslash I^j \cup I^j \backslash I^i)$ et $\lambda_{spec}(s_c, I^x) = O^k$ **then**
29: **if** $i_k = i_{k+1}$ et $\exists s_d \in S_M$ qui vérifie $\delta_{spec}(s, I^x) = s_d$ et $\lambda_{spec}(s_d, i_k) = o_{k+1}$ **then**
30: $k = k + 2$
31: $s = \delta_{spec}(s_d, i_k)$
32: Retour à la ligne 4.
33: **else**
34: L'implantation n'est pas conforme à la spécification
35: **end if**
36: **else**
37: L'implantation n'est pas conforme à la spécification
38: **end if**
39: **end if**
40: **end while**

4.3.3 Exemple

L'algorithme de construction du modèle du système en boucle fermée lorsque l'observation est réalisée aux bornes de l'API est illustré via un API dont le modèle de spécification est donné par la Figure 4.6. On suppose que le programme de commande implanté dans le contrôleur est celui décrit dans la section 4.2.3.1. L'observation des entrées et des sorties se faisant cette fois-ci non pas dans le contrôleur mais à ses bornes, le chronogramme présenté Figure 4.19 décrit les valeurs observées pour 10 pas d'observation avec une situation initiale où l'état 1 est actif et où seule la variable Pf est à $Vrai$. Pour rappel, les valeurs des entrées et sorties à l'intérieur de l'API pour la même période d'observation sont données en Figure 4.7.

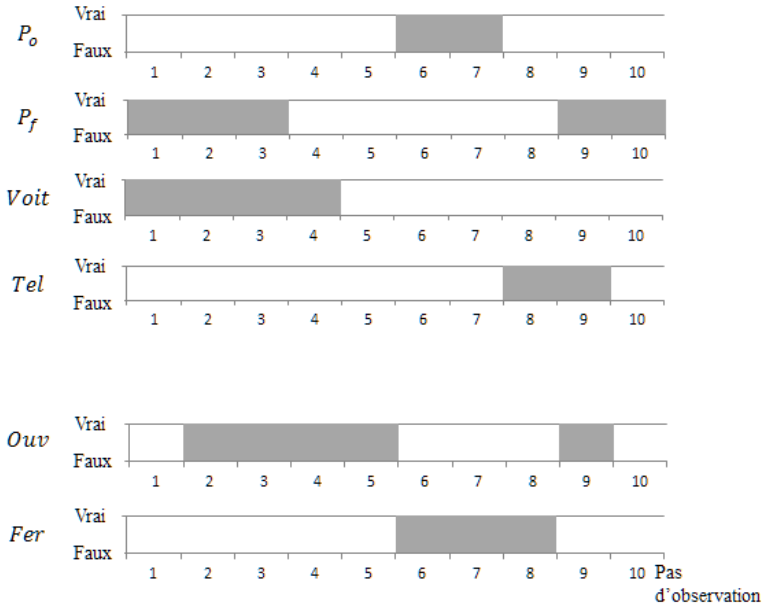


Figure 4.19 – Chronogramme des valeurs des variables d'entrée et de sortie observées aux bornes de l'API

Au premier pas d'observation, la combinaison de valeurs des variables d'entrée est $\overline{Po}.\,Pf.\,Voit.\,\overline{Tel}$ et la combinaison de valeurs de variables de sortie est $\overline{Ouv}.\overline{Fer}$. Depuis l'état 1 du modèle de spécification, aucune transition n'est étiquetée selon ces valeurs de variables d'entrée et de sortie. Le test en ligne 5 de l'algorithme est donc négatif et la suite du programme est exécutée à partir de la ligne 17 afin de détecter un éventuel phénomène de synchronisation ou de désynchronisation. Le test de retard de la ligne 17 est bien vérifié puisque les valeurs de sorties observées au pas 1 correspondent à la self-loop étiquetée par $\overline{Po}.Pf.\overline{Voit}.\overline{Tel}$. Le pas d'observation suivant est donc pris en compte pour la suite de l'analyse. Celui-ci est composé du couple ($\overline{Po}.Pf.Voit.\overline{Tel}$, $Ouv.\overline{Fer}$).

Les valeurs des variables d'entrée étant identiques au pas précédent, un phénomène de synchronisation est à proscrire. L'algorithme passe alors à la ligne 28. La valeur des variables de sortie observées correspond à une transition dans le modèle de spécification étiquetée par les valeurs de variable d'entrée observées. Au pas d'observation suivant, le couple ($\overline{Po}.Pf.Voit.\overline{Tel}$, $Ouv.\overline{Fer}$) est une nouvelle fois observé, validant le test réalisé à la ligne 29. Le phénomène de retard à la lecture à $Vrai$ de la variable $Voit$ (visible par comparaison avec le chronogramme de la Figure 4.7) est donc détecté et la phase de construction du modèle du système en boucle fermée peut être poursuivie à partir de l'état 3.

Les pas d'observation 4 à 7 sont ensuite analysés par l'algorithme qui détecte un comportement valide à chaque pas et construit les transitions associées dans le modèle du système en boucle fermée.

Le pas 8 d'observation est composé du couple ($\overline{Po}.\overline{Pf}.\overline{Voit}.Tel$, $\overline{Ouv}.Fer$) qui ne correspond pas à une transition existante partant de l'état 2 actif du modèle de spécification. Le test de la ligne 17 de l'algorithme détecte un éventuel phénomène de retard à l'émission des sorties. Le pas d'observation 9 est alors ($\overline{Po}.Pf.\overline{Voit}.Tel$, $Ouv.\overline{Fer}$). Il comprend bien un changement de valeurs des variables d'entrée et correspond au franchissement d'une transition du modèle de spécification. Le résultat du test de la ligne 19 de l'algorithme est donc positif et le phénomène de synchronisation des évènements sur les variables Tel et Pf (visible par comparaison avec le chronogramme de la Figure 4.7) est correctement détecté.

4.4 Synthèse

Ce chapitre propose donc une technique de validation d'un contrôleur logique couplé à sa partie opérative. Le développement de cette technique a conduit à deux contributions distinctes. Tout d'abord, la mise en place d'un critère permettant de déterminer l'avancée et surtout la fin de la phase d'observation du système en boucle fermée, puis la définition d'un algorithme de construction et de validation du modèle du système en boucle fermée en fonction de la technique d'observation employée.

Le premier apport est nécessaire car la taille du modèle du système en boucle fermée à valider est inconnue du fait de l'absence de connaissance sur la partie opérative. Une

solution est donc proposée, utilisant la connaissance du modèle de spécification, afin de compter et déterminer le nombre maximal de transitions du modèle du système en boucle fermée. Cette connaissance est essentielle pour garantir la validité du contrôleur sur la totalité de son comportement observable en un temps donné lorsqu'il est couplé à sa partie opérative.

Le second apport est un algorithme d'identification en boite grise lorsque un sur-ensemble de l'espace d'état est connu. Il se présente en deux parties. Tout d'abord, un premier algorithme est défini afin de construire le modèle du système en boucle fermée lorsque l'observation est réalisée à l'intérieur du contrôleur. Ce type d'observation des variables d'entrée et de sortie limite les phénomènes à prendre en compte mais nécessite d'utiliser des techniques d'observation invasives pour le contrôleur. Dans un second temps, un nouvel algorithme est défini permettant de construire le modèle du système en boucle fermée lorsque l'observation est réalisée aux bornes de l'API. Ce nouvel algorithme permet une observation non-invasive et est capable de détecter des lectures synchrones (resp. asynchrones) d'évènements asynchrones (resp. synchrones).

Conclusions et perspectives

Conclusions

Les différentes contributions réalisées durant ces travaux de thèse ont été présentées dans ce mémoire. Trois contributions peuvent être identifiées dans le cadre de la formalisation de modèle de spécification, du test de conformité, et de la validation en boucle fermée. Les deux dernières contributions servent au développement de techniques complémentaires pour la validation fonctionnelle de contrôleurs logiques.

La première contribution concerne la construction, à partir de Grafcet, du modèle formel de spécification. Des travaux déjà existants permettent de modéliser sous la forme d'une machine de Mealy un Grafcet interprété avec recherche de stabilité. Cependant, cet algorithme d'interprétation n'est pas celui retenu dans les applications industrielles. En effet, **l'interprétation sans recherche de stabilité est préférée pour ses résultats en terme de réactivité**. Ce mémoire propose donc une extension à la construction de modèle formel pour le cas où le Grafcet est implanté sans recherche de stabilité. Il est en effet essentiel de construire le bon modèle de la spécification qui servira de modèle de référence aux techniques de validation pour avoir un verdict fiable sur la validité du contrôleur logique.

La seconde contribution prend la forme de la définition d'un ensemble de relations de conformité pour le test de conformité. Ces relations de conformité sont adaptées au test de contrôleurs logiques, et non pas à des modèles de leur programme de commande. En effet, que l'interprétation du modèle de spécification Grafcet soit avec ou sans recherche de stabilité, la solution pour la relation de conformité proposée permet de tester des contrôleurs logiques en **évitant l'émission de verdicts biaisés tout en garantissant l'objectif de test**. Ses principales caractéristiques sont alors :

- d'être basée sur des observations de séquences de valeurs de variables de sortie pour chaque pas de test,

- d'être capable de traiter des séquences MIC sans erreur de verdict.

La dernière contribution présentée dans ce mémoire est une technique de validation de contrôleur logique lorsque celui-ci est couplé à sa partie opérative, réelle ou simulée. Cette technique est une alternative au test de conformité permettant de valider le contrôleur en configuration d'utilisation, c'est-à-dire couplé à une partie opérative, et sur le seul comportement possible en système bouclé. Les travaux sur le développement de cette technique ont nécessité la levée de deux verrous :

- être capable de déterminer **à partir de quel moment le comportement du système bouclé a été suffisamment observé** pour pouvoir rendre un verdict sur la validité du contrôleur,

- être capable d'analyser les séquences d'entrée/sortie observées et de distinguer les non-conformités des phénomènes dûs à la technologie de lecture des entrées/sorties du contrôleur ou encore de l'observateur.

Perspectives

Afin d'élargir le spectre d'application des méthodes proposées, plusieurs perspectives peuvent être envisagées. Ces perspectives peuvent être classées en trois grandes catégories.

Une première perspective est la levée de l'hypothèse de distingabilité des états du modèle formel par observation des valeurs des variables de sortie. Cette hypothèse a été posée sur le modèle formel afin d'être capable de discerner les fautes de transfert présentée en Figure 1.12. En effet, pour détecter le fait que le modèle de comportement du programme de commande possède une transition qui n'a pas le bon état aval, il faut être capable de distinguer chacun des états et cela ne peut se faire que par la seule information observable : la valeur des variables de sortie. Cependant, cette hypothèse est très contraignante pour le modèle de spécification Grafcet dont est issu la machine de Mealy. En effet, avoir des états de la machine de Mealy distingables par observation des sorties signifie que le modèle Grafcet ne présente pas deux situations différentes qui émettent le même ensemble de sorties. Une piste de recherche pour assouplir cette hypothèse serait de définir une notion de k-distingabilité, telle qu'il soit toujours possible

de distinguer deux états par observation des variables de sortie après un franchissement de k transitions.

L'ajout de la prise en compte de propriétés temporisées est aussi une perspective importante. En effet, les milieux industriels usent abondamment de temporisations que ce soit par exemple pour ajouter des sécurités ou pallier à un manque de capteurs. Actuellement, l'absence de la prise en compte de ces temporisations se traduit par un indéterminisme sur les modèles de spécification qui ne sont donc plus traitables par les méthodes développées ici. Cependant, plusieurs axes du projet ANR VACSIM dans lequel s'inscrivent ces travaux ont mis en évidence les difficultés qui apparaissent en terme de complexité et d'explosion combinatoire dès que le temps est introduit dans les modèles. Une solution simple mais invasive peut cependant être introduite pour prendre en compte les temporisations du Grafcet en créant des variables internes vues comme des variables d'entrée annonçant la fin de la temporisation.

Enfin, le sujet a été évoqué mais il serait intéressant d'approfondir le déroulement de l'opération de validation en boucle fermée lorsque la partie opérative est simulée. Il faut en effet être capable de garantir que le modèle de la partie opérative est exempt de fautes. De plus, avoir une partie opérative simulée peut permettre de simuler les divers temps d'attente entre deux évènements d'entrée et accélérer la convergence de la courbe des transitions franchies. Développer des règles de variations sur les durées des opérations de la partie opérative permettrait ainsi de nettement améliorer l'efficacité de la validation en boucle fermée. Un autre axe de recherche peut aussi traiter de la validation du contrôleur en présence de fautes de la partie opérative en simulant celles-ci.

Liste des notations

A_G	Ensemble des actions du Grafcet
A_C	Ensemble des actions continues
A_S	Ensemble des actions mémorisées
$C(V_I)$	Condition de stabilité pour une localité
et	Pas de test défini dans les travaux précédents
$E(V_I)$	Condition de franchissement d'une évolution d'un automate des localités
$Evol$	Ensemble des évolution de l'automate des localités
$FC(V_I, X_G)$	Réceptivité associée à une transition
$F_t(V_I)$	Condition de franchissement d'une transition dans une situation donnée
$F_\tau(V_I)$	Condition de franchissement d'un ensemble de transitions τ
I_{AL}	Alphabet d'entrée de l'automate des localités
I_M	Alphabet d'entrée de la machine de Mealy
L	Ensemble des localités de l'automate des localités
L_{boucle}	Langage accepté par le modèle du système bouclé
$l_D \in L$	Ensemble des localités aval d'une évolution d'un automate des localités
l_{Init}	Localité initiale de l'automate des localités
L_{impl}	Langage accepté par le modèle de comportement du programme de commande implanté dans le contrôleur logique
L_p	Langage accepté par le modèle de la partie opérative

$l_U \in L$	Ensemble des localités amont d'une évolution d'un automate des localités
MIC	Multiple-Input-Change ; évènements synchrones sur plusieurs variables d'entrée
M_{impl}	Modèle de comportement du programme de commande implanté dans le contrôleur sous la forme d'une machine de Mealy
M_{spec}	Modèle de la spécification à implanter sous la forme d'une machine de Mealy
O_{AL}	Alphabet de sortie de l'automate des localités
O_{Em}	Ensemble des sorties émises pour une localité
O_M	Alphabet de sortie de la machine de Mealy
S_{Act}	Ensemble des étapes actives du Grafcet
S_D	Ensemble des étapes aval d'une transition
SFT	Ensemble de transitions simultanément franchissables
S_G	Ensemble des étapes du Grafcet
SIC	Single-Input-Change ; évènement sur une seule variable d'entrée
s_{init}	Etat initial de la machine de Mealy
S_{Init}	Etape initiale du Grafcet
S_M	Ensemble des états de la machine de Mealy
$SSFT$	Ensemble des sous-ensembles de transitions simultanément franchissables
S_U	Ensemble des étapes amont d'une transition
T_G	Ensemble des transitions du Grafcet
TS	Séquence de test définie dans les travaux précédents
$\mathcal{TS}$	Nouvelle séquence de test composée uniquement des variables d'entrée
T_{valid}	Ensemble des transitions validées pour une situation donnée
V_I	Ensemble des variables d'entrée du contrôleur logique
V_O	Ensemble des variables de sortie du contrôleur logique
$\delta_M : I_M \times S_M \rightarrow S_M$	Fonction de transition de la machine de Mealy

$\lambda_M : I_M \times S_M \to O_M$	Fonction de sortie de la machine de Mealy

Bibliographie

Bacic, M. (2005). On hardware-in-the-loop simulation. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, 3194–3198.

Bank, J. (2010). *Discrete Event System Simulation*. Pearson.

Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen (2001). *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer.

Bierel, E., Douchin, O., and Lhoste, P. (1997). Grafcet : from theory to implementation. *Journal Européen des Systèmes Automatisés*, 31(3), 543–559.

Boehm, B.W. (1979). *Classics in software engineering*. Yourdon Press, Upper Saddle River, NJ, USA, yourdon, edward nash edition.

Brinksma, E. and Tretmans, J. (2001). Testing transition systems : An annotated bibliography. In F. Cassez, C. Jard, B. Rozoy, and M. Ryan (eds.), *Modeling and Verification of Parallel Processes*, volume 2067 of *Lecture Notes in Computer Science*, 187–195. Springer Berlin Heidelberg.

Bullock, D., Johnson, B., Wells, R.B., Kyte, M., and Li, Z. (2004). Hardware-in-the-loop simulation. *Transportation Research Part C : Emerging Technologies*, 12(1), 73 – 89.

Cabasino, M., Giua, A., and Seatzu, C. (2007). Identification of petri nets from knowledge of their language. *Discrete Event Dynamic Systems*, 17(4), 447–474.

Chow, T. (1978). Testing software design modeled by finite-state machines. *Software Engineering, IEEE Transactions on*, SE-4(3), 178–187.

Clarke, Jr., E.M., Grumberg, O., and Peled, D.A. (1999). *Model Checking*. MIT Press, Cambridge, MA, USA.

David, R. and Alla, H. (2007). *Du Grafcet aux réseaux de Pétri*. Traité des Nouvelles Technologies. Hermès, France, 2e éd. rev. augm. edition.

Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A.R., and Yevtushenko, N. (2010). Fsm-based conformance testing methods : A survey annotated with experimental evaluation. *Inf. Softw. Technol.*, 52(12), 1286–1297.

Dotoli, M., Fanti, M.P., and Mangini, A.M. (2008). Real time identification of discrete event systems using petri nets. *Automatica*, 44(5), 1209–1219.

Estrada-Vargas, A.P., López-Mellado, E., and Lesage, J.J. (2010). A Comparative Analysis of Recent Identification Approaches for Discrete-Event Systems. *Mathematical Problems in Engineering*, 2010. 21 pages.

Estrada-Vargas, A.P., Lopez-Mellado, E., and Lesage, J.J. (2013). Automated Modelling of Reactive Discrete Event Systems from External Behavioural Data. In *Proc. of CONIELECOMP'13*, pp. 120–125. Cholula Puebla, Mexique.

Fabian, M. and Hellgren, A. (1998). PLC-based implementation of supervisory control for discrete event systems. In *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, volume 3, 3305–3310 vol.3.

Giua, A. and Seatzu, C. (2005). Identification of free-labeled petri nets via integer programming. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, 7639–7644.

Guignard, A. and Faure, J.M. (2013a). Construction de modèles formels de contrôleurs logiques pour le test de conformité. In *Actes des 5èmes Journées Doctorales / Journées Nationales MACS (JD-JN-MACS'13)*, 00. Strasbourg, France.

Guignard, A. and Faure, J.M. (2013b). Enforcing I/O sequences for PLC validation purposes. In *IEEE Emerging Technologies and Factory Automation*, 00. CAGLIARI, Italie.

Guignard, A. and Faure, J.M. (2014a). A conformance relation for model-based testing of plc. In *12th International IFAC-IEEE Workshop on Discrete Event Systems (WODES'14)*, 00. Cachan, France.

Guignard, A. and Faure, J.M. (2014b). Validation of logic controllers from events observation in a closed-loop system. In *IEEE Emerging Technologies and Factory Automation*, 00. BARCELONA, Spain.

Guignard, A. and Faure, J.M. (2013c). Formal models for conformance test of programmable logic controllers. *Journal Européen des Systèmes Automatisés*, 47(4-8), pp. 423–446.

IEC 1012 (2004). *IEEE Standard for Software Verification and Validation*. Institute of Electrical and Electronics Engineers.

IEC 60848 (2002). *GRAFCET Specification language for sequential function charts (2nd ed.)*. International Electrotechnical Commission.

IEC 61131-1..IEC 61131-8 (2000-2010). *Programmable controllers Part 1 to 8*. International Electrotechnical Commission.

IEC 61131-2 (2007). *Programmable controllers - Part 2 : Equipment requirements and tests*. International Electrotechnical Commission.

Isermann, R., Schaffnit, J., and Sinsel, S. (1999). Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, 7(5), 643 – 653.

Klein, S., Litz, L., and Lesage, J.J. (2005). Fault detection of Discrete Event Systems using an identification approach. In *16th IFAC world Congress*. Praha, Czech Republic.

Kwan, M.K. (1962). Graphic programming using odd or even points. *Chinese Math*, 1(273-277), 110.

Lee, D. and Yannakakis, M. (1996). Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8), pp. 1090–1123.

Lhoste, P., Panetto, H., and Roesch, M. (1993). Grafcet : from syntax to semantics. *Automatique Productique Informatique Industrielle*, 27(1), 127–141. Special issue "Advances in Grafcet", ISSN 0296-1598.

Li, M. and Kumar, R. (2012). Model-based automatic test generation for simulink/stateflow using extended finite automaton. In *Automation Science and Engineering (CASE), 2012 IEEE International Conference on*, 857–862.

Lu, B., Wu, X., Figueroa, H., and Monti, A. (2007). A low-cost real-time hardware-in-the-loop testing approach of power electronics controls. *Industrial Electronics, IEEE Transactions on*, 54(2), 919–931.

Machado, Jose, J., Denis, B., and Lesage, J.J. (2006). Formal Verification of Industrial Controllers : with or without a Plant model ? In *7th Portuguese Conference on Automatic Control, CONTROLO'06*. Lisboa, Portugal.

McMillan, K. (1993). *Symbolic Model Checking*. Springer US.

Meda, M.E., Ramirez, A., and Malo, A. (1998). Identification in discrete event systems. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, volume 1, 740–745 vol.1.

Naito, S. and Tsunoyama, M. (1981). Fault detection for sequential machines by transitions tours. In *Proceedings of the IEEE fault tolerant computer symposium*, 238–243.

Patterson, F.J. (2009). Systems engineering life cycles : Life cycles for research, development, test, and evaluation ; acquisition ; and planning and marketing. *Handbook of systems engineering and management*, pp. 65–78.

Ponce de Leon, H., Haar, S., and Longuet, D. (2012). Conformance relations for labeled event structures. In A. Brucker and J. Julliand (eds.), *Tests and Proofs*, volume 7305 of *Lecture Notes in Computer Science*, 83–98. Springer Berlin Heidelberg.

Provost, J., Roussel, J.M., and Faure, J.M. (2014). Generation of single input change test sequences for conformance test of programmable logic controllers. *Industrial Informatics, IEEE Transactions on*, 10(3), 1696–1704.

Provost, J. (2011). *Test de conformité de contrôleurs logiques spécifiés en grafcet*. Ph.D. thesis, École normale supérieure de Cachan.

Provost, J., Roussel, J.M., and Faure, J.M. (2010). SIC-testability of sequential logic controllers. In *10th International Workshop on Discrete Event Systems (WODES'10)*, 203–208.

Provost, J., Roussel, J.M., and Faure, J.M. (2011a). Testing programmable logic controllers from finite state machines specification. In *Dependable Control of Discrete Systems (DCDS), 2011 3rd International Workshop on*, 1–6. IEEE.

Provost, J., Roussel, J.M., and Faure, J.M. (2011b). Translating Grafcet specifications into Mealy machines for conformance test purposes. *Control Engineering Practice*, 19(9), pp. 947–957.

Roth, M., Lesage, J., and Litz, L. (2010). Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems. In *American Control Conference (ACC)*, 2601–2606.

Schlager, M. (2008). *Hardware-in-the-Loop Simulation*. VDM Verlag, Saarbrücken, Germany, Germany.

Tretmans, J. (1996). Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3), 103–120.

Zhu, H. and He, X. (2002). A methodology of testing high-level petri nets. *Information and Software Technology*, 44(8), 473 – 489.

Annexe : A propos de la génération et de l'exécution de test pour une interprétation sans recherche de stabilité

Une fois le modèle obtenu sous la forme d'une machine de Mealy, une séquence de test est construite puis appliquée aux bornes de l'API afin de réaliser l'opération de validation de cet API. La séquence est construite pas à pas jusqu'à ce que la totalité des transitions de la machine de Mealy aient été franchies au moins une fois.

Cependant, il apparait que lorsque l'implantation est réalisée avec un algorithme d'exécution sans recherche de stabilité plusieurs séquences de sorties différentes peuvent être observées pour une même séquence d'évènements d'entrée alors que la machine de Mealy est déterministe. Ceci est illustré à l'aide de la machine de Mealy générique partielle présentée en Figure 5.1.

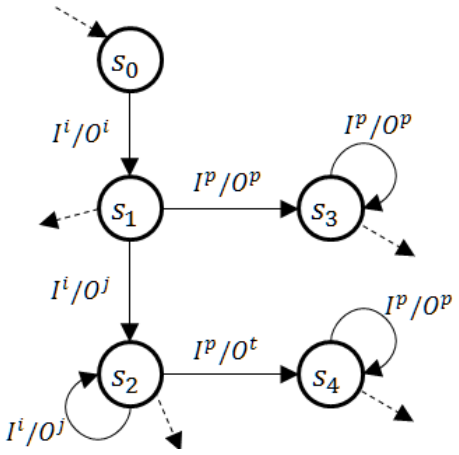


Figure 5.1 – Machine de Mealy partielle présentant deux transitions successives franchissables sans changement de valeur des variables d'entrée

En supposant l'état s_0 actif et en appliquant successivement les combinaisons de variables d'entrée I^i puis I^p, on peut observer successivement les combinaisons de variables de sortie (O^i, O^p) ou (O^i, O^j, O^t) selon que la combinaison d'entrée I^i a été appliquée au contrôleur pendant un ou plus d'un cycles API.

En considérant un programme de commande modélisable par la machine de Mealy donnée en Figure 5.2 et pour une même séquence de test donnée par la relation (5.1) :

$$\mathcal{TS}_{part} = ((a.b), (a.\bar{b}), (\bar{a}.\bar{b})) \tag{5.1}$$

et en faisant varier le nombre de cycles API que dure chaque pas de test, on obtient :

Si chaque pas de test dure un unique cycle API :

$$\sigma_{Obs1} = ((O_1.O_2), (\overline{O_1}.\overline{O_2}), (\overline{O_1}.\overline{O_2})) \tag{5.2}$$

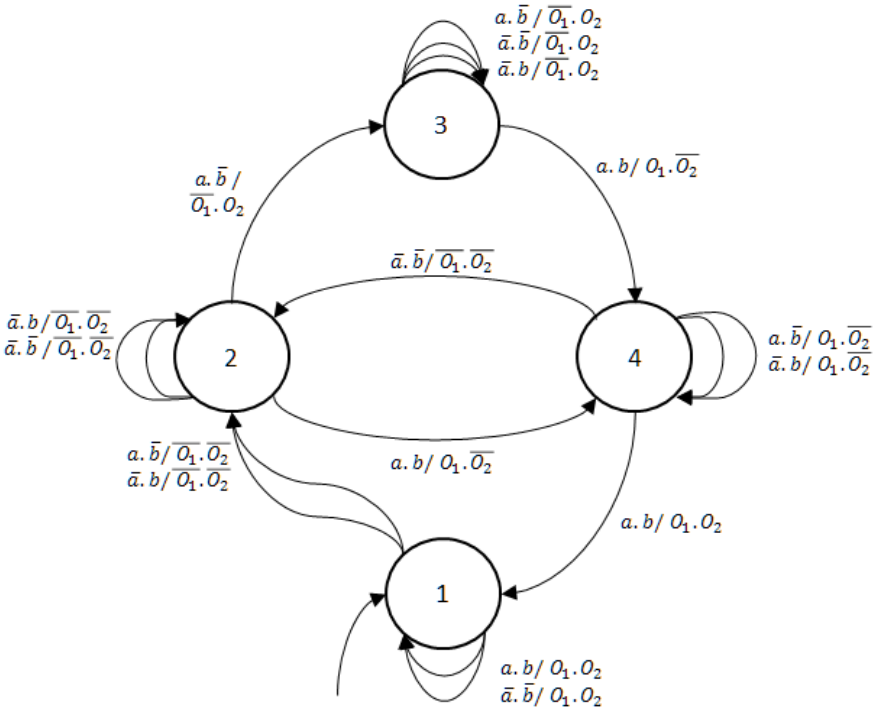


Figure 5.2 – Rappel de la machine de Mealy correspondant à une interprétation sans recherche de stabilité présentée en Figure 2.8

Si chaque pas de test dure deux cycles API :

$$\sigma_{Obs1} = ((O_1.O_2),(O_1.O_2),(\overline{O_1}.\overline{O_2}),(\overline{O_1}.O_2),(\overline{O_1}.O_2),(\overline{O_1}.O_2)) \tag{5.3}$$

La première séquence d'observation, donnée par la relation (5.2), correspond à un une durée de pas de test égale à un cycle API. Ainsi, depuis l'état stable 1, le second pas de test entraine le franchissement de la transition vers l'état 2, puis le changement de variables d'entrée au cycle API suivant entraine le franchissement de la transition vers l'état 3.

La seconde séquence d'observation, donnée par la relation (5.3), montre qu'augmenter la durée d'un pas de test (ici le pas 2) ne conduit pas nécessairement à la duplication des valeurs des variables de sortie observées. En effet, après avoir atteint l'état 2, la valeur des variables d'entrée du second pas de test entraine aussi le franchissement de la transition menant à l'état 4. Le changement suivant de valeur des variables d'entrée correspondant au troisième pas de test entraine le franchissement de la self-loop sur l'état 4 et les valeurs des variables de sortie restent inchangées.

Cela signifie que pour parcourir le premier chemin il faut garantir de pouvoir ne passer qu'un cycle API à chaque pas de test. Hors, cela requiert de synchroniser l'envoi

de la séquence de test avec la lecture des entrées du contrôleur et ce n'est pas possible lors d'une exécution du test non-invasive. La séquence de test définie par la relation (5.1) qui attend en sortie la séquence d'observation σ_{Obs1} n'est donc pas exécutable car on ne peut pas imposer le changement de valeur des variables d'entrée à un cycle API précis.

Par conséquent, cette séquence de test ne doit pas être générée telle quelle. Plus généralement, après un changement de valeur des variables d'entrée, il faut attendre d'avoir atteint un nouvel état stable, au sens où la transition sortant de cet état étiquetée selon la valeur des variables d'entrée considérée est une self-loop, pour pouvoir exécuter le prochain pas de la séquence de test. Cela revient alors à construire la séquence de test à partir de l'automate des localités stables présenté Figure 2.4 dont le principe de construction est justement de franchir toutes les transitions possibles jusqu'à stabilité.

La séquence de test devient alors :

$$\mathcal{TS} = (a.b, a.\overline{b}, \overline{a}.\overline{b}, \overline{a}.b, a.b, \overline{a}.\overline{b}, \overline{a}.b, \overline{a}.\overline{b}, a.\overline{b}, a.b, \overline{a}.b, a.b) \tag{5.4}$$

Bien entendu, comme l'implantation est réalisée avec un algorithme d'exécution sans recherche de stabilité, c'est ce modèle qui doit être utilisé lors de l'exécution du test pour fournir le verdict. **Cela signifie donc que les modèles pour la construction de la séquence de test et pour l'analyse de la séquence d'observation sont différents.**

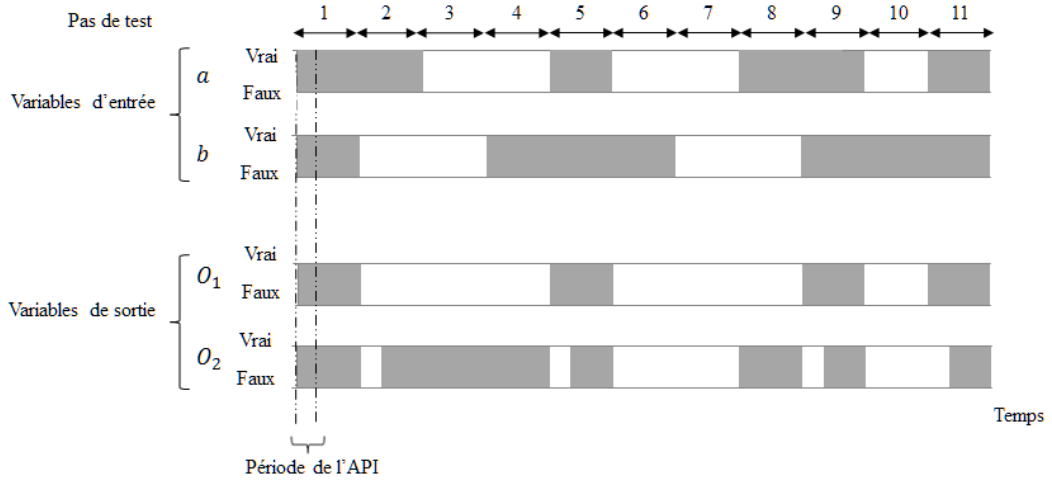


Figure 5.3 – Chronogramme présentant les résultats de l'exécution de la séquence de test donnée en relation (5.4)

Ainsi, l'exécution de cette séquence de test et les sorties observées pour une im-

plantation conforme sont présentées dans la Figure 5.3. Chaque pas de test doit être appliqué suffisamment longtemps pour que les pas de test liés à une évolution fugace, tous les franchissement successifs de transition du Grafcet, puis la self-loop associée à la situation stable soient visibles. Ici, une durée de 3 cycles API est nécessaire.

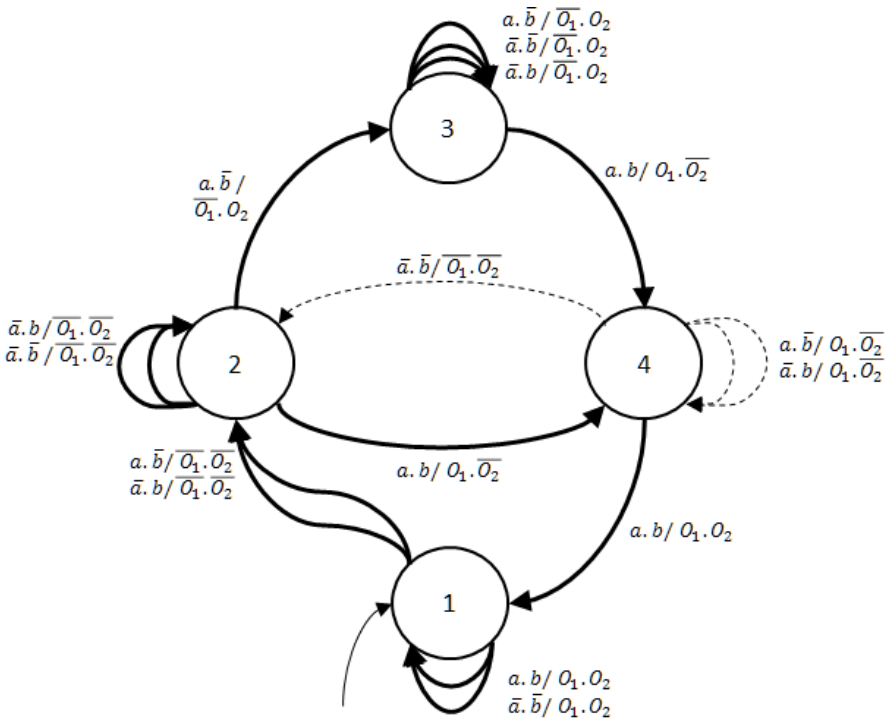


Figure 5.4 – Transitions (en gras) franchies lors de l'exécution de la séquence de test donnée en relation (5.4)

La Figure 5.4 met en évidence les transitions franchies correspondant aux séquences d'observation (en gras sur la figure). On remarque que 3 transitions ne sont jamais testées, cela signifie donc que l'objectif de test commun, qui est que toutes les transitions de la machine de Mealy doivent être testées au moins une fois, n'est pas respecté.

Ceci est dû au fait que le seul moyen d'atteindre l'état 4 est via une évolution fugace. Tester les self-loop ainsi que les transitions sortantes autres que celles impliquées dans les évolutions fugaces nécessite donc de changer de valeur des variables d'entrée lorsque l'état 4 est actif. Selon les contraintes techniques présentées précédemment, il n'est pas possible d'interrompre une évolution fugace et ces transitions ne sont donc pas testables sans mise en place d'un protocole expérimental invasif pour l'API. Un nouvel objectif, cohérent avec le protocole d'exécution de la séquence de test doit être défini. Celui-ci peut être énoncé comme : Toute transition testable[4] de la machine de Mealy doit être

testée au moins une fois.

Pour résumer, il faut noter que lorsque le programme de commande implanté dans le contrôleur logique à tester est spécifié par un Grafcet interprété sans recherche de stabilité :

- La séquence de test est toujours construite à partir de la machine de Mealy déduite de l'ALS, et donc modélisant une interprétation avec recherche de stabilité
- L'analyse de la séquence d'observation après exécution de cette séquence de test doit, elle, être réalisée en prenant comme modèle de spécification la machine de Mealy déduite de l'AL, et donc en prenant une relation de conformité adaptée à ce type d'interprétation du Grafcet.

4. i.e. l'état amont de cette transition existe aussi dans l'ALS